PROVIDING COMPUTER TELEPHONY

TO DISABLED USERS WITH

VOICE RECOGNITION

BY

NABIL AMINE NABHAN

Associate of Science
Oklahoma State University
Stillwater, Oklahoma
1987

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
1988

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
1991

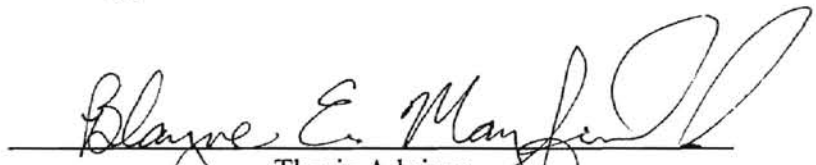
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1997

PROVIDING COMPUTER TELEPHONY

TO DISABLED USERS WITH

VOICE RECOGNITION

Thesis Approved:



Thesis Advisor

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to Dr. Blayne E. Mayfield for his intelligent supervision, constructive guidance, inspiration, friendship, assistance, and ongoing encouragement in guiding the completion of this work. My sincere appreciation extends to my other committee members Dr. Jacques LaFrance and Dr. Lu Huizu, whose guidance, assistance, encouragement, and friendship are also invaluable.

More over, I wish to express my thanks to Mr. David Harrison, all my friends specially Abbas Yassine, and teachers that I have met during my college years, and to my coworkers at the Windows Technology Group.

I would like to give my special appreciation to my parents, my three lovely sisters and my lovely wife Soulafa for their support, encouragement at times of difficulty, love and understanding throughout this whole process. To them, my newborn son Adam and to my deceased grandparents and two Uncles I dedicate this thesis and extend my deepest appreciation and love.

Above all, I thank God.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

NOMENCLATURE

ASR	Automatic Speech Recognition.
ADC	Analog to Digital Converter.
Back-End	Microsoft Access Database.
CT	Computer Telephony.
CTI	Computer Telephony Integration.
dB	Decibel.
DASR	Dependent Automatic Speech Recognition.
DAC	Digital to Analog Converter.
DLL	Dynamic Link Library.
DOS	Disk Operating System.
DSVD	Digital Simultaneous Voice and Data.
Front-End	Windows based user interface.
GUI	Graphical User Interface.
IASR	Independent Automatic Speech Recognition.
ISA	Industry Standard Architecture.
IVR	Interactive Voice Recognition.
MS	Microsoft.
OLE	Object Linking and Embedding.

OS	Operating System.
PBX	Private Branch Exchange.
PC	Personal Computer.
PnP	Plug And Play.
TAPI	Telephony Application Programming Interface.
TSR	Terminate and Stay Resident.
UI	User Interface.

CHAPTER I

INTRODUCTION

Computer Telephony (CT) system for disabled users is one of the newest, most useful, sophisticated, and reliable tools of modern civilization. In a very complex and technological world, CT provides a relatively simple link between normal and disabled users and the rest of the world [HAR89], making the telephones on their desk obsolete [ELG96].

Speech processing is an important element of modern communication, which has received much of the attention as a new type of interface between users and computers [EJPJ87].

In this thesis I designed a modest voice controlled CT system with a full-duplex speakerphone for hands free operation. This system allows disabled users, especially the blind, to command the computer to make a phone call to anyone anywhere with the use of their own voice. This will enable them to use the CT system without the use of a keyboard or a pcint and click device (mouse) which are more often a curse than a blessing.

Voice input commands are used as an input interface to control the phone system, and a Text-To-Speech synthesis tool used as an audio output messaging utility.

CHAPTER II

CURRENT SYSTEMS

Disabled users, especially the blind, who would like to reach out and make a phone call anytime for whatever reason specially in emergency situations [CARL92] cannot use the current CT systems with the modern window-based user interfaces, such as the keyboard and point and click device (mouse) which are not user friendly to disabled users, especially the blind. The usage of these interfaces is difficult and frustrating for disabled users, especially the blind. These barriers distance disabled users, especially the blind from using computers to fulfill their needs [VAS95]. These obstacles have raised many challenging design issues.

Using a regular phone to make a call is very difficult if not impossible for disabled users, especially the blind [DOUG90].

Currently, some of the existing systems [MSI95] that run under DOS or Windows that are designed to allow disabled users, especially the blind, to make a phone call are limited to the phone numbers per person that can be stored in the database, they run as a TSR under DOS, reducing the system's resources and memory and do not support multitasking. They are expensive and are not designed with either voice-recognition as an input interface, a Text-To-Speech synthesis tool for audio messaging, do not convey

the characteristics of the different aspects of the interface in an efficient non-visual form, or full-duplex speakerphone for hands free operation.

2.1 Microsystems software HandiPHONE

HandiPHONE is a DOS and windows 3.x based system computer controlled hands-free telephone system by Microsystems Software [MSI95]. It is a Terminate and Stay Resident program (TSR) in memory if running under DOS that allows the user of an IBM or compatible PC to access any standard Tip-Ring or Private Branch Exchange (PBX) telephone system via a speakerphone or handset without the need for any manual intervention.

2.2 Microsoft Phone

Microsoft (MS) Phone [ELG96] is a Windows 95 based system by Microsoft aimed at pushing the PC and telephone even closer together. The product lets the user dial a phone and review voice- and e-mail messages with speech commands.

2.3 Scope

This system's implementation is limited to a Microsoft commercial Phone control application, voice recognition engine, and Text-To-Speech tool. MS Phone (See Figure 8 Page 29) system is used as a control application, the voice recognition engine is the medium used to take the voice command inputs, and the Text-To-Speech tool is used to produce the audio output. However, the details of the process of how MS Phone (See Figure 8 Page 29) makes the call, MS Voice recognizes the input voice commands, and

the process of converting text into speech is beyond the scope of this thesis; good reviews are found in the Microsoft press releases [MSPCD96].

2.4 Automatic Speech Recognition (ASR)

ASR is how machines understand spoken input. The spoken input usually comes from humans. Speech recognition systems have been available for over twenty years, however the technology has evolved more slowly than predicted. Today, thousands of these systems are in operation and the number of new systems is growing rapidly. Before detailing the design of this system, one needs to understand the capabilities of speech recognition technology. Today's technology includes Discrete-, Connected-, and Continuous-Word recognizers and Speaker-Dependent and -Independent Recognizers.

2.4.1 Discrete-, Connected-, Continuous-Word Recognizers

Speech recognizers may contain the following types of word recognition. Speech recognition engines that recognize a series of spoken words where more than 250 milliseconds of silence separate each word are classified as Discrete-Word Recognizers. Speech recognition engines that recognizes a series of spoken words where at least 50 but not more than 250 milliseconds of silence separate each word are classified as Connected-Word Recognizers. . Speech recognition engines that recognizes a series of spoken words where less than 50 milliseconds of silence separates each word are classified as Continuous-Word Recognizers.

2.4.2 Dependent and Independent Automatic Speech Recognition

Speech recognizers are either dependent or independent. Dependent Automatic Speech Recognizers (DASR) identifies spoken input words after it has been trained for individual human voices. All users must train DASR before using them. This training is done in an appropriate environment(s) and it involves choosing a vocabulary where each word is repeated several times in order to form a reference template. This type of speech recognizers limits the number of users. On the other hand, Independent Automatic Speech Recognizers (IASR) provides high recognition accuracy under different environmental conditions, regardless of sex, dialect, and other speaker characteristics. This type of recognizers requires no training and minimizes PC memory usage and requirements because a single reference template would accommodate multiple users.

2.5 Speech Synthesis

Speech synthesis technology has been one of the major factors that have influenced non-visual user interface adaptations. Synthesized speech technology nowadays is one of the most powerful tools that is helping with closing the gap between blind users and new computers built with high signal-processing chips and fast inexpensive memory resulting in high speech quality [OMA90] [LEO91] [LAZ93]. As a result, the majority of systems designed for blind users are utilizing this technology as the principle format of communicating the visual user interface.

Previous commercial applications produced speech output by concatenating a sequence of prerecorded words, syllables, or phonemes which are the smallest units of speech that distinguish one word sound from the other [WOO94]; usually found in

dictionaries to dictate how words are pronounced, and are the elements on which computer speech is based. These applications had their disadvantages including a limit on the words that could be produced to what had been previously recorded. The way it was recorded determined the way it sounded when played back. These constraints have limited the flexibility of stored voice response systems and have affected the efficiency of the auditory user interface.

Text-To-Speech synthesis systems, on the other hand, operate directly from an input text or data stream to produce real time, understandable speech instead of using prerecorded human speech. These systems take arbitrary text as input together with optional user specified commands to control the system parameters such as phrasing and rate to produce real-time synthetic speech. The process of converting text into speech parameters involves advanced information processing. Text-To-Speech has the advantage of producing an unlimited vocabulary over the old way of producing speech by the use of prerecorded human speech that had a vocabulary limitation. On the contrary, Speech synthesis disadvantage is that it produces an unnatural sound.

Current speech synthesizers provide means for the user to control the number of words spoken per minute, simulate a variety or a combination of male/female voices, and they link with the computer Operating System (OS) so that the visual interface and input devices become verbally interactive and controlled by the user.

CHAPTER III

DESIGN CONSIDERATION FOR COMPUTER TELEPHONY

There are some hardware and software considerations which must be taken into account to achieve the goal of closing the gap between blind users and computers by making hardware and software cheaper, compatible, compliant, and standardized by all software and hardware vendors to make available systems more effective tools for blind users.

3.1 Design Constraints

Conveying the characteristics of the different aspects of the interface in an efficient non-visual form has not been an easy task. The use of Text-To-Speech synthesis tool in this system gave it a high information bandwidth, but it was not as high as the visual interface. Thus, the non-visual user interface has to be less complex and capable of achieving as many of the benefits in the visual user interface [EMR94].

Users with visual impairments have limited real-time access to computer information; thus, time has been one of the most valuable commodities. Short-term memory and strong concentration have also been important since blind users need the means available to sighted people to check the screen and refresh their memories [EDW89]. Consequently, a highly relevant design issue for telephony systems with voice

recognition has been that feedback must be brief, yet informative. System installation, setup, procedures, and tasks that are performed by sighted users would require an enormous effort by blind users to accomplish. For example, a sighted user can determine the location of the cursor or mouse-pointer in a client area of an application by glancing at the monitor. On the contrary, the system must be designed in a way that once an object gets focus the system should read the name of the object by the use of a Text-To-Speech synthesis tool.

To achieve a reasonable similar perception of the running application by a blind user as that of a sighted user, any system must be designed to keep the visual and non-visual interfaces coherent [JUB92].

3.1.1 Analog and Digital phone systems

Analog and digital signals are incompatible, MS Phone (See Figure 8 Page 29) needs an analog phone line (the kind installed in homes) and will not work on a digital line (the kind used at the offices with a PBX system).

3.1.2 Cost

The Computer Telephony Integration (CTI) hardware and software market is enormous and is just evolving. Telephony is finally making inroads on the Personal Computer (PC). Not only it is expensive, but also it is hard to find the software and the hardware that will work together.

3.1.3 Current user interfaces

Navigation data entry and manipulation can be communicated by a keyboard, mouse, or any combination. However, this has added to the complexity of the solution since the user has to remember mouse clicks and/or key to press and their location.

3.1.3 System installation and setup

System installation and setup are not an easy task and would require an enormous effort to be performed by disabled users, especially the blind. For example, setting MS Phones' dialing properties (See Figure 9 Page 30).

3.2 Solutions to the above constraints

The problem with the type of phone line signal (analog or digital) could be overcome by the use of a Digital-To-Analog (DAC) and an Analog-To-Digital Converter (ADC) (See Figure 1 Page 10) or special devices to do a two-way signal processing and conversion between the analog phone blaster and the phone line. Since the cost of hardware and software is dropping daily, it looks promising that soon a nice CT system will become affordable, if not free like the MS Phone (See Figure 8 Page 29) which will mainstream computer telephony and will be integrated into Windows 95 itself as part of the explorer interface, as a free applet or both within the next two years. IASR is used to overcome the navigation and data entry and manipulation. Regretfully, at this time system installation and setup must be done by a sighted user.

Figure 1. Analog-To-Digital and Digital-To-Analog converter

3.3 Screen Readers Major Control Facilities

One of the most important responsibilities of the system is to convey the windows user interface activity to the blind user by translating the graphical interface into words.

3.3.1 Verbalizing the Current Cursor or Mouse Pointer Location

A critical and important function of the system is to provide the blind user with the cursor location. In order for the phone system to convey the GUI activity, it has to verbalize the name of the object that has focus so that the blind user can tell where is the action [THA93]. The location of the cursor or mouse pointer is usually the active object that has focus. Focus is changed by issuing the system the "Press TAB", "Press SHIFT-TAB", "Next", or "Previous" voice commands which would simulate pressing the TAB or SHIFT-TAB keys on the Keyboard. The user interface object's name is verbalized by passing its name to a Voice-Text function every time the location of the cursor.

3.3.2 Automatic Monitoring of Errors and Information Messages Feature

The phone system is programmed to pass the string expression of errors, warnings, action-status, or other type of messages to a Text-To-Speech synthesis function to inform or alert the blind user of certain system actions such as deleting a person's list of phones. Run time errors are handled differently in a way such that not only is the blind user alerted to contact a software engineer but also the error is saved into an error log text

file with a date-time stamp for the engineer to review upon contact. This helps to lift the burden of the blind user in remembering the error and when it happened.

3.3.2.1 Auditory Cues

Text-To-Speech synthesis is not the only way to communicate the User Interface (UI) information. Technology has been carried out to use auditory cues [EDW89] such as one or more audible beeps that are associated with certain actions or could be used to convey a specific message depending on the number of audible beeps.

CHAPER IV

SYSTEM IMPLEMENTATION

The solution proposed in this project is a user friendly Windows 95 based CT speakerphone system designed with speech-recognition as an input interface to allow disabled users, especially the blind, to use their own voice as an input interface to command the computer to call anyone anywhere with a simple command such as "John Doe - home - Call or Dial." The blind user needs to operate the system (See Figure 6 Page 28) by going through the following steps. The user would command the computer to set focus on the Name ComboBox by issuing the "Name" voice command. The person's name is selected by spelling out his or her name using preprogrammed words that stands for every letter of the alphabet (See Table I Page 85). The verbalized letters are pickup by the microphone and entered by the system into the Name ComboBox. The same previous step is followed to select the phone's physical location, but the "Location" voice command is used to set focus on the location ComboBox. The user would issue the voice command "Get Phone" (See Table III Page 89) to retrieve the phone number from a database, whose only limitation is storage space available. If the phone number does not exist in the database, the computer will display a visual message and/or verbalize a string expression text prompting the user to enter the phone number in any way that is most convenient for him/her. Finally the "Dial" voice command is issued by the user to command the system to activate MS Phone (See Figure 8 Page 29) to dial the phone

number. If the person's name and the phone's physical location are in the database and the phone number is not, then the system will prompt the user in two ways to enter the phone number that is added to the database before the number is dialed. The "Hang up" voice command is used to release the phone line and activate the phone system. Otherwise the system will prompt the user to enter the above-required information to process the call. This process is shown in Figure 2 Page 13.

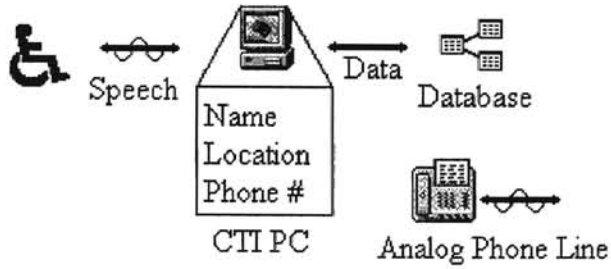


Figure 2. System implementation process control

The design of the phone system was implemented to allow disabled users, especially the blind, to use their own voice to command the computer to make a phone call. The system not only was designed specifically for blind users but also for normal users. Disabled and normal users are becoming physically equal because of our advanced technology, so the system was designed to be able to be used with a mouse, a keyboard, and/or voice commands to control the system. A Text-To-Speech synthesis tool is used to convert words in a character string into speech played over the computer speakers. The character string is place into the playback queue after it is passed as a parameter to the Speak method of the Voice-Text object. The exposed OLE automation Voice-Text object is accessed and controlled through Visual Basic, allowing it to be used for communicating with disabled users, especially the blind. The system was designed to work with a variety of multimedia and telephony devices (voice modems) that are

Telephony Application Programming Interface (TAPI) compliant and built with full-duplex speakerphone for hands free operation.

A 32-bit Visual Basic development kit was used to design the Front-End screens running under a 32-bit multitasking Microsoft Windows 95 that allows the disabled user to work on something else while the MS Phone (See Figure 8 Page 29) is constantly checking the communication port for incoming calls. An Access Back-End was used to create a database of three tables (See Figure 3 Page 15), a Name table consisting of two fields (NameID, and NameText) used for storing the name of the person, a Location table consisting of two fields (LocationID, and LocationText) used for storing the phone's physical location, and a Phone table consisting of four fields (PhoneID, NameID, LocationID, and PhoneNum) used for storing phone numbers associated with each person ID and the phone's physical location ID. A 32-bit low-level Dynamic Link Library (DLL) consisting of functions written in C\C++ programming languages called Telephony Application Programming Interface (TAPI32.DLL), was called from within Visual basic to interface with a TAPI compliant multimedia and telephony device (voice modem) that is designed with a full-duplex speakerphone capability for hands free operation.

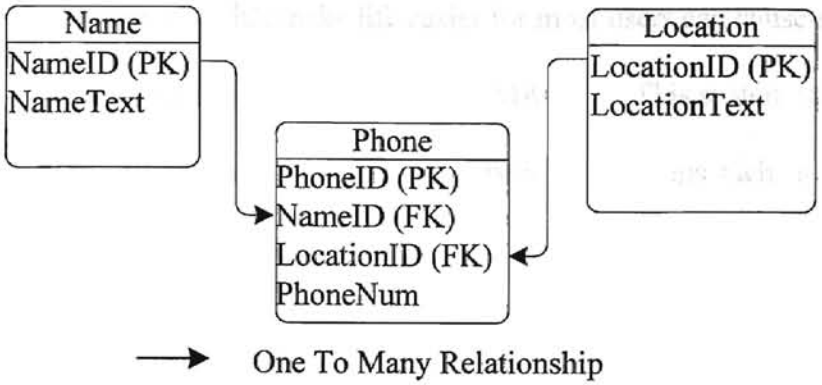


Figure 3. Back-End Access Database

The system will execute the tapiRequestMakeCall function to requests a voice call between the user and a remote party specified by its phone number. The request is made to TAPI, which passes it to an MS Phone (See Figure 8 Page 29) application that is registered as a recipient of such requests. If the call control application is not running, TAPI32.DLL will attempt to launch the highest-priority call control application listed for RequestMakeCall in TELEPHON.INI file [SWA95].

The system was designed to be controlled in one or any combination of three input interfaces. An independent voice recognition, a point and click device (mouse), or a keyboard. The system uses preprogrammed voice command words that stands for Arabic numbers and English alphabets to translate voice commands into either mouse clicks or keyboard strokes.

The phone system uses Microsoft Voice speech recognition engine to process voice commands. Microsoft Voice was designed by using a 32-bit low level speech Dynamic Link Library (SPEECH.DLL).

The very technologies that make life easier for most users can cause problems for those with physical handicaps such as poor vision [MAC96]. This system utilizes current technologies to solve the problem for those with physical handicaps such as poor vision.

4.1 Independent Automatic Voice Recognition and the Software interface

Voice input commands are received by the microphone that is connected to Creative lab's Phone Blaster. The phone blaster passes these commands to Microsoft Voice engine for signal processing to be recognized, if successful the associated subroutine, function, or code with that action is executed as if the command was carried out by using a keyboard or a mouse.

4.2 The Software and the database interface

The phone number of the selected person and the phone's physical location is retrieved from the database when a "Get Phone" command is issued. If the phone number exists it is passed as a parameter to the tapiRequestMakeCall API function to request a voice call between the user and a remote party by issuing the "Dial" command. The request is made to TAPI, which passes it to an MS Phone (See Figure 8 Page 29) application that is registered as a recipient of such requests. If the call control application is not running, TAPI32.DLL will attempt to launch the highest-priority call control application listed for RequestMakeCall in TELEPHON.INI file.

4.3 The Software and the Hardware interface

MS Phone (See Figure 8 Page 29) is launched or activated in a full-duplex speaker mode using Creative Lab's Phone Blaster built in 16-bit sound card that is connected to monitor built in speakers used for voice output and a microphone used to pick up voice input commands. The phone number is dialed using the TAPI compliant Creative lab's Phone Blaster to communicate with the analog phone line as if a regular phone is being used. Once the phone conversation is over the user would command MS Phone (See Figure 8 Page 29) to "Hang up" to release the phone line and "BlindPhone" to activate this project's system.

4.4 Voice commands format

Three voice command tables are provided by MS Voice and the third was designed for the phone system. These tables are used to fully control the phone system (See Figure 6 Page 28) and MS Phone (See Figure 8 Page 29). A list of the tables follows:

Table I Page 85 - is a list of preprogrammed words that stand for every letter of the English alphabet and the Arabic 0 to 9 numbers.

Table II Page 87 - is a list of other Keyboard keys.

Table III Page 89 - is a list of commands used by the phone system.

4.5 Microsoft speech synthesis tool

Implementing Voice Text in an application involves the use of the Voice-Text object. The object manages the interaction between the Text-To-Speech engine that creates the

speech and the computer speakers that serves as the destination for the speech. Before an application can begin using Voice Text, it must create an instance of the Voice-Text object and register itself with the object. The Voice-Text object is exposed as an OLE automation object, allowing it to be accessed and controlled through Visual Basic. Words in a character string are converted into speech played over the computer speakers. These words are verbalized by Microsoft's speech synthesis depending on the case of the word. The word is verbalized one letter at a time if all are capital letters. Otherwise the word is fully pronounced by the tool. A list of the VoiceText Class Methods and Properties are shown in (Figure 4 Page 18).

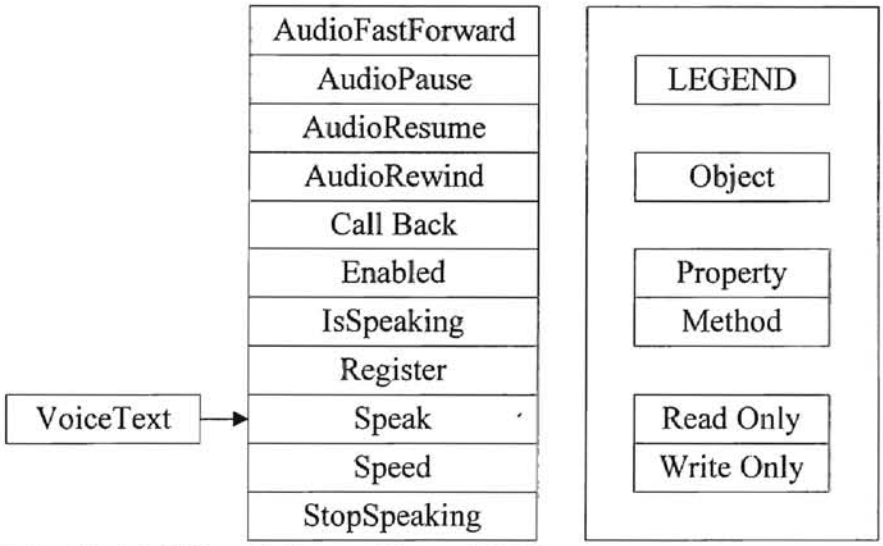


Figure 4. VoiceText 1.0 Type Library - Extended View

CHAPTER V

SUMMARY AND RECOMMENDATIONS

A variety of federal and state legislative actions, not the least of which is the Americans with Disabilities act, combined with public sentiment, are resulting in increasing awareness and emphasis on accessibility. In concert with this movement, the software industry has been asked to make its products more accessible to users with disabilities. This has raised questions among the members of the industry as to what exactly the problems are, and which steps they can take to help make their products more accessible [VAN92]. This thesis focuses on a small subset in an effort to narrow this gap for users who are blind or severely visually impaired by customizing access to a voice controlled telephony application.

5.1 Summary

The possibilities for disabled users, especially the blind those uncomfortable or unskilled with mice or keyboards, and especially Windows users are endless. IVR allows disabled users, especially the blind to make phone calls without having to dial the number physically or to pick up the handset. Once connected, disabled users, especially the blind, could speak into the microphone and listen through the PC's speakers.

5.2 Recommendations for Future work

The user interface for a blind user is unfortunately hard and not as advanced as that of a sighted user. There are several improvements that can be made to enhance the non-visual and non-physical interfaces provided in this research.

5.2.1 Wireless Headset

The interface could further be improved through the use of an Anti-Noise Computer headset that uses a technology that effectively cancels background noise and improves microphone sound quality, eliminating speaker-microphone feedback.

5.2.2 Recognition Accuracy and Dictation

Commercially available speech recognizers exhibit a wide range of performance. To access performance, recognition accuracy must be defined and measured. Recognition accuracy refers to the percentage of the time the recognizer correctly classifies an utterance.

A voice recognizer can make substitution, rejection, or spurious response errors. The most critical error is a substitution error. A substitution error occurs when the recognizer substitutes an incorrect word for a spoken word. A rejection error occurs when the recognizer does not classify a spoken word, but reject it. A spurious response error occurs when the recognizer classifies a sound or invalid word as a valid word. Both substitution and rejection error rates determine a recognizer's accuracy. Substitution errors could be minimized by reducing the background noise. Even though MS Voice makes some substitution errors, recognition accuracy close to 96% was calculated when

the system was tested in an office with a background noise around 3 decibels (dB) and improved when the background noise was minimized (See Figure 5 Page 21).

The interface could further be improved through the use of a dictation tool that allows the user to dictate directly into data entry fields.

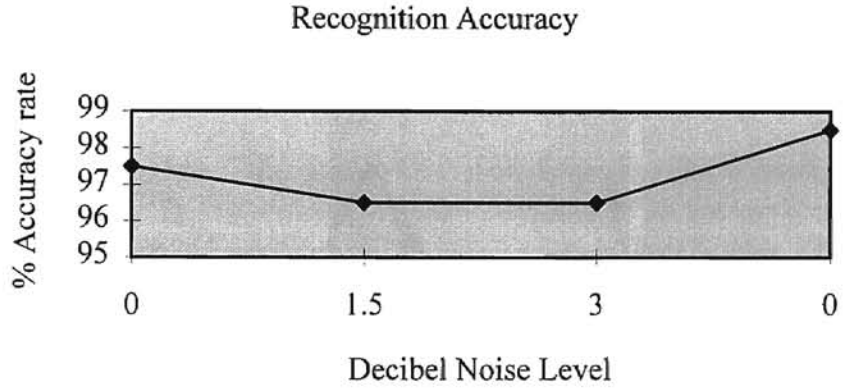


Figure 5. Voice recognition accuracy

5.2.2 Design and Quality Assurance

Designing and quality assurance are a very difficult phases for a software engineer to fully design and test such a system without feedback from a disabled users, especially the blind. One must fulfill these steps to insure user satisfaction of a software system of this significance to disabled users, especially the blind to achieve a robust, fully functional, and user friendly system.

BIBLIOGRAPHY

[BRO92] C. Brown. "Assistive Technology Computers and Persons with Disabilities." Communications of the ACM, Vol.35, No.5, pp.36-45, May 1992.

[EDW88] A. Edwards. "The Design of Auditory Interface for Visually Disabled Users." In Proceedings of ACM Conference On Human Factors in Computing Systems, CHI'88, New York, pp.83-88, May 1988.

[EDW89] A. Edwards. "Soundtrack: An Auditory Interface for Blind Users." Human Computer Interaction, Vol.4, pp.45-66, 1989.

[EDW94] W. Edwards, E. Mynatt, T. Rodrigues. A Non-Visual Interface to the X Windows System. Mercator Project, Georgia Institute Of Technology, 1994.

[ELG96] Mike, Elgan. "The telephone on your desk is obsolete. I am not kidding." Windows magazine, Vol.7, No.2, pp.53-56, Feb. 1996.

[FOS93] Peter Foster, Dr. Thomas B. Schalk, Speech Recognition, Telecom Library, Inc. 1993.

[GIL87] Gilden, D. "A robotic hand as a communication aid for the deaf-blind." Proceedings of the Twentieth Hawaii International Conference on System Sciences, Vol.3, pp.264-273, 1987.

[GLI92] Ephraim P. Glinert. "Computers and People with Disabilities." Communications of the ACM, Vol.35, No.5, pp.33-35, May 1992.

[GRI90] D. Griffith. "Computer Access for Persons Who are Blind or Visually Impaired: Human Factors Issues." Human Factors, Vol.32, No.4, pp.467-475, Aug. 1990.

[HAL95] Aouni Hallal. "Providing Text-Mode Access To Blind Users With An Application Using Text-To-Speech Synthesis." Master Of Science Thesis, May 1995.

[HAR89] M. Harb. Modern Telephony, Prentice Hall, Englewood Cliffs, NJ, 1989.

[HAS96] David, Haskin. "Kurzweil Voice Gets Your PC To Listen Up." PC Magazine, Vol.15, No.14, pp.58, Aug. 1996.

[HOL88] J. N. Holmes. Speech Synthesis and Recognition, Van Nostrand Reinhold, UK, 1988.

[JUB92] Paul Jubinski. "Virtac, a Virtual Tactile Computer Display." In the IEEE Proceedings of the Johns Hopkins National Search for Computing Applications to Assist Persons with Disabilities, Maryland, pp. 208-211, Feb., 1992.

[KOE95] Scott Koegler. "Speaking your mind." INFOWORLD, Vol.17, Issue 48, pp.92-109, Nov. 1995.

[LAZ93] J. Lazzaro. Adaptive Technologies For Learning and Work Environments. American Library Association, 1993.

[LAZ94] Joe Lazzaro. "Adapting GUI software for the blind is no easy task." Byte, Vol.19, No.5, pp.33, May 1994.

[LEO91] M. Leonard. "Speech Poised To Join Man-Machine Interface." Electronic Design, pp.43-48, Sep. 1991.

[LIN96] David S. Linthicum. "Make Voice Response Sing." Byte, pp. 53-56, May 1996.

[LUD90] L. Ludwig. "Extending The Notion of a Window System to Audio." Computer, pp.66-72, August 1990.

[LUH96] Luhmann, Rick & Grigonis, Richard. "Booming CT Call Control." Computer Telephony, Vol.5, Issue 7, pp.34-96, July 1996.

[MAC96] Machrone, Bill. "Disabling technology." PC Magazine, Vol.14, No.16, pp.83, Sept. 1995.

[MAR96] Margulies, Ed. "Creative Labs' New Phone Blaster." Computer Telephony, Vol.5, Issue 7, pp.28-29, July 1996.

[MSI95] Microsystems Software, Inc. Product information package, "HandiPhone." pp. 7, Jun. 1995.

[MSPCD96] Microsoft Press Bookshelf CD for Windows 95, Microsoft, Inc. 1996.

[OMA90] M. O'Malley. "Text-To-Speech Conversion Technology." Computer, pp.17-23, Aug. 1990.

[OPP92] C. Opperman. "Application of Smart Screen Technology In Text-Based Voice Output, Screen Access Programs." In Proceedings of the Seventh Annual Conference on Technology and Persons with Disabilities, Los Angeles, California, pp.391-395, March 1992.

[QUI91] Justin, Quillinan. "Headstart for the Danes?" Telecom World, pp. 8-11, Sep. 1991.

[SWA95] Richard K. Swadley. Visual Basic 4 Unleashed, SAMS Publishing, Indianapolis, IN, 1995.

[THA93] Jim Thatcher. "The Problems and Challenges of the Graphical User Interface." Braille Monitor, pp.9-16, November 1993.

[VAS95] Vaspori, T., Arato, A. "Ten years of computer use by visually impaired people in Hungary." Journal of Microcomputer Applications, Vol.18, No.4, pp.313-317, UK, Oct 1995.

[VAN92] G. Vanderheiden. Making Software More Accessible for People with Disabilities. Release 1.2, Trace Center, University of Wisconsin, 1992.

[WOO94] JoAnne Woodcock, Computer Dictionary, Microsoft Press, 2nd Edition. 1994.

[YAN87] E.J. Yannakoudakis, P.J. Hutton. Speech Synthesis and Recognition Systems, Ellis HorWood Limited, New York, NY, 1987.

APPENDIXES

APPENDIX A

HARDWARE AND SOFTWARE REQUIREMENTS

A user has to install the required hardware and software on the workstation.

1. Machine Requirements

   A. IBM 486DX/33 100% compatible or more powerful system with 8M of memory.

   B. Microsoft TAPI compliant Data, Fax, Voice, and DSVD Modem designed with a full-duplex speakerphone capability (Creative Labs Phone Blaster).

   C. Full-length 16-bit Industry Standard Architecture (ISA) slot for a Creative labs Sound Blaster 16-bit with Creative Soundo'LE feature.

   D. 16M of hard disk space for the audio card's software and phone system.

   E. An Electret Condenser, Uni-directional Mićrophone.

   F. Any 8Ω Audio Output Speakers.

   G. EGA or VGA card (VGA recommended)

2. Program requirements

   A. A 32-bit Windows 95 or NT Operating System (OS).

   B. Microsoft Phone for a full range of telephone functions.

   C. Microsoft Voice for independent automatic speech recognition.

D. Microsoft Text-to-Speech synthesis tool.

APPENDIX B

USER INTERFACE WINDOWS

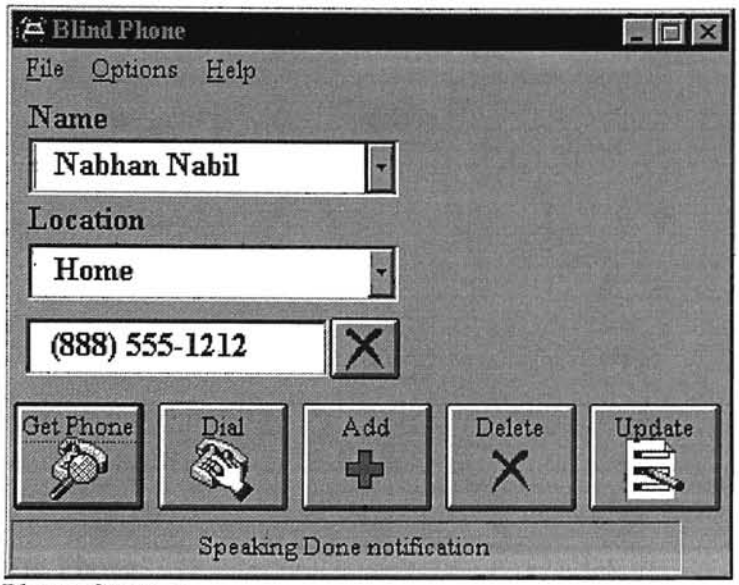


Figure 6. Blind Phone System

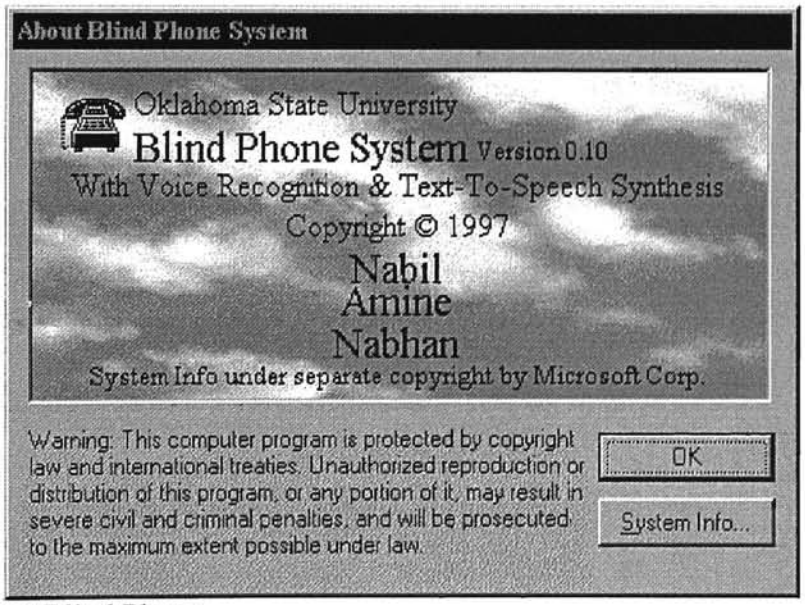


Figure 7. About Blind Phone

Figure 8. Microsoft Phone

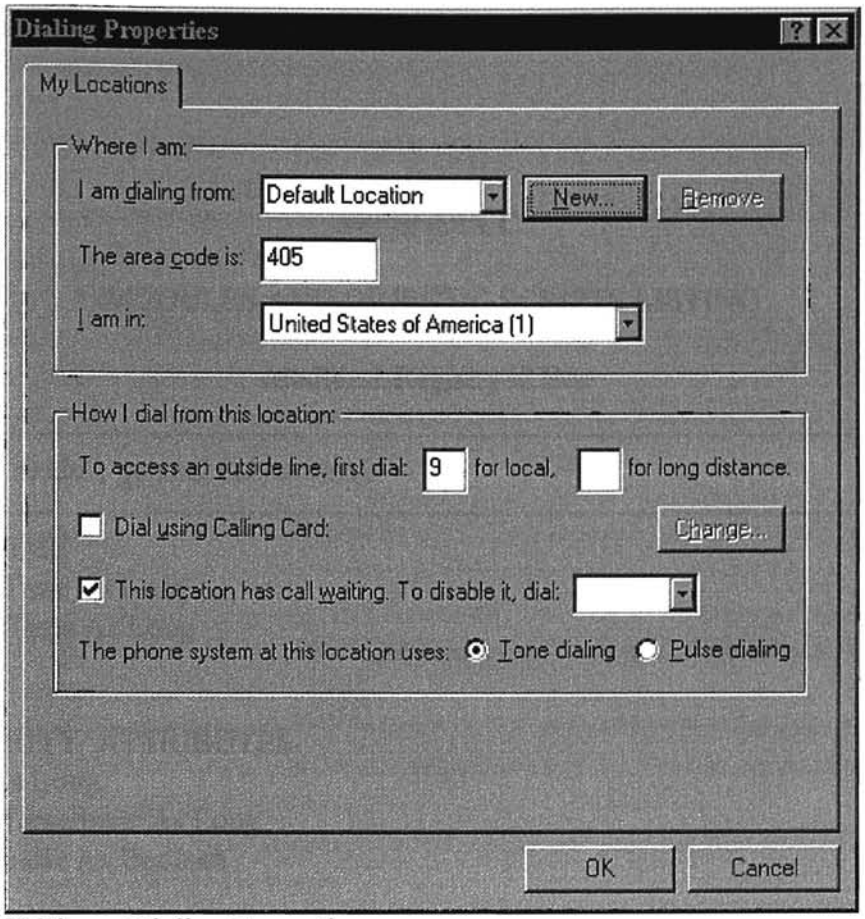


Figure 9. MS Phone Dialing Properties

APPENDIX C

MODULES AND OBJECTS' SCRIPTS LISTING

Module 1 Registry32.Bas

```
'-------------------------------------------------------------------------------------------------
' Registry32 module general declarations.
'-------------------------------------------------------------------------------------------------
Type FILETIME
  lLowDateTime As Long
  lHighDateTime As Long
End Type

Type SECURITY_ATTRIBUTES
  nLength As Long
  lpSecurityDescriptor As Long
  bInheritHandle As Boolean
End Type

' Data for the RegQueryInfo call
Type QUERYINFOSTRUCTURE
  hkey As Long
  Class As String
  ClassLen As Long
  Reserved As Long
  KeyCnt As Long
  MaxKey As Long
  MaxClass As Long
  Values As Long
  MaxValue As Long
  MaxData As Long
  Security As Long
  LastWrite As FILETIME
End Type

Public kInfo As QUERYINFOSTRUCTURE

' Data for the RegEnumValue call
Type QUERYVALUESTRUCTURE
```

```
  Item As String
  ItemValue As String
  ItemType As Long
End Type

Public vInfo() As QUERYVALUESTRUCTURE

' 32-bit Registry APIs
Declare Function RegDeleteKey Lib "advapi32.dll" Alias "RegDeleteKeyA" (ByVal
hkey As Long, ByVal lpSubKey As String) As Long
Declare Function RegDeleteValue Lib "advapi32.dll" Alias "RegDeleteValueA" (ByVal
hkey As Long, ByVal lpValueName As String) As Long
Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyExA" (ByVal
hkey&, ByVal lpszSubKey$, dwOptions&, ByVal samDesired&, lpHKey&) As Long
Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hkey&) As Long
Declare Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQueryValueExA"
(ByVal hkey&, ByVal lpszValueName$, ByVal lpdwRes&, lpdwType&, ByVal
lpDataBuff$, nSize&) As Long
Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA" (ByVal
hkey&, ByVal lpszValueName$, ByVal dwRes&, ByVal dwType&, lpDataBuff As Any,
ByVal nSize&) As Long
Declare Function RegConnectRegistry Lib "advapi32.dll" (ByVal lpMachineName$,
ByVal hkey&, phkResult&) As Long
Declare Function RegCreateKeyEx Lib "advapi32.dll" Alias "RegCreateKeyExA"
(ByVal hkey&, ByVal lpSubKey$, ByVal Reserved&, ByVal lpClass$, ByVal
dwOptions&, ByVal samDesired&, lpSecurityAttributes&, phkResult&,
lpdwDisposition&) As Long
Declare Function RegFlushKey Lib "advapi32.dll" (ByVal hkey&) As Long
Declare Function RegEnumKeyEx Lib "advapi32.dll" Alias "RegEnumKeyExA" (ByVal
hkey&, ByVal dwIndex&, ByVal lpName$, lpcbName&, ByVal lpReserved&, ByVal
lpClass$, lpcbClass&, lpftLastWriteTime As FILETIME) As Long
Declare Function RegEnumValue Lib "advapi32.dll" Alias "RegEnumValueA" (ByVal
hkey&, ByVal dwIndex&, ByVal lpName$, lpcbName&, ByVal lpReserved&,
lpdwType&, lpValue As Any, lpcbValue&) As Long
Declare Function RegQueryInfoKey Lib "advapi32.dll" Alias "RegQueryInfoKeyA"
(ByVal hkey&, ByVal lpClass$, lpcbClass&, ByVal lpReserved&, lpcSubKeys&,
lpcbMaxSubKeyLen&, lpcbMaxClassLen&, lpcValues&, lpcbMaxValueNameLen&,
lpcbMaxValueLen&, lpcbSecurityDescriptor&, lpftLastWriteTime As FILETIME) As
Long

' Registry return codes
Public Const ErrorSuccess As Long = 0
Public Const ErrorBadDB As Long = 1009
Public Const ErrorBadKey As Long = 1010
Public Const ErrorCantOpen As Long = 1011
```

```
Public Const ErrorCantRead As Long = 1012
Public Const ErrorCantWrite As Long = 1013
Public Const ErrorOutOfMemory As Long = 14
Public Const ErrorInvalidParameter As Long = 87
Public Const ErrorAccessDenied As Long = 5
Public Const ErrorNoMoreItems As Long = 259
Public Const ErrorMoreData As Long = 234
Public Const ErrorFileNotFound As Long = 2

' Additional return codes
Public Const ErrorUnsupportedType As Long = 7000
Public Const ErrorKeyAlreadyExists As Long = 7001

' System-defined registry keys
Public Const HkeyClassesRoot = &H80000000
Public Const HkeyCurrentUser = &H80000001
Public Const HkeyLocalMachine = &H80000002
Public Const HkeyUsers = &H80000003
Public Const HkeyPerformanceData = &H80000004
Public Const HkeyCurrentConfig = &H80000005
Public Const HkeyDynData = &H80000006

' Registry value constants
Public Const RegNone As Long = 0
Public Const RegSz As Long = 1
Public Const RegExpandSz As Long = 2
Public Const RegBinary As Long = 3
Public Const RegDWord As Long = 4
Public Const RegDWordLittleEndian As Long = 4
Public Const RegDWordBigEndian As Long = 5
Public Const RegLink As Long = 6
Public Const RegMultiSz As Long = 7
Public Const RegResourceList As Long = 8
Public Const RegFullResourceDescriptor As Long = 9
Public Const RegResourceRequirementsList As Long = 10

' Constants for creating new keys
Public Const RegOptionNonVolatile = 0
Public Const RegOptionVolatile = 1
Public Const RegCreatedNewKey = &H1
Public Const RegOpenedExistingKey = &H2

' Registry permissions
Public Const KeyQueryValue = &H1&
Public Const KeySetValue = &H2&
```

```
Public Const KeyCreateSubKey = &H4&
Public Const KeyEnumerateSubKeys = &H8&
Public Const KeyNotify = &H10&
Public Const KeyCreateLink = &H20&
Public Const ReadControl = &H20000
Public Const WriteDAC = &H40000
Public Const WriteOwner = &H80000
Public Const Synchronize = &H100000
Public Const StandardRightsRequired = &HF0000
Public Const StandardRightsRead = ReadControl
Public Const StandardRightsWrite = ReadControl
Public Const StandardRightsExecute = ReadControl
Public Const StandardRightsAll = &H1F0000
Public Const KeyRead = StandardRightsRead Or KeyQueryValue Or
KeyEnumerateSubKeys Or KeyNotify
Public Const KeyWrite = StandardRightsWrite Or KeySetValue Or KeyCreateSubKey
Public Const KeyExecute = KeyRead
Public Const KeyAllAccess = ((StandardRightsAll Or KeyQueryValue Or KeySetValue
Or KeyCreateSubKey Or KeyEnumerateSubKeys Or KeyNotify Or KeyCreateLink) And
(Not Synchronize))

Public Function Reg32KeyExists(ByVal vsKey As String) As Long
  '-----------------------------------------------------------------------------------------------
  ' Determines whether the specified key exists in the registry. key$ is a fully qualified
Registry key
  ' with subkeys separated by a backslash (\). Returns ErrorSuccess (0) if the key exists in
the registry
  ' and ErrorFileNotFound (2) if it does not, otherwise returns whatever error was
generated by the Win32
  ' Registry API calls.
  '-----------------------------------------------------------------------------------------------
  Dim hkey As Long
  Dim lMainKey As Long
  Dim sSubKey As String
  Dim lReturnCode As Long

  ' Parse out the main key and sub key info
  lReturnCode = ParseKeys(vsKey, lMainKey, sSubKey)
  If lReturnCode <> ErrorSuccess Then
    Reg32KeyExists = lReturnCode
    Exit Function
  End If

  ' Open the key with read access
  lReturnCode = RegOpenKeyEx(lMainKey, sSubKey, 0&, KeyRead, hkey)
```

```
  Reg32KeyExists = lReturnCode

  ' If it opened, be sure to close it
  If lReturnCode = ErrorSuccess Then
    lReturnCode = RegCloseKey(hkey)
  End If
End Function

Public Function Reg32CreateNewKey(ByVal vsKey As String) As Long
  '-----------------------------------------------------------------------------------------------
  ' Create a new key in the registry if it does not already exist. key$ is a fully qualified
Registry key
  ' with subkeys separated by a backslash (\). Will create multiple nested keys in a
  ' single call if they are specified in Key$ and do not already exist. For example, If no
subkeys exist
  ' under HKEY_CURRENT_USER then calling this routine with
"HKEY_CURRENT_USER\MyKey\
  ' MySubKey" will create the key "MyKey" under HKEY_CURRENT_USER and
"MySubKey" under
  ' "MyKey". Returns ErrorSuccess (0) if the key did not previously exist or
ErrorKeyAlreadyExists
  ' (7001) if the subkey was already in the registry, otherwise returns whatever error was
generated by
  ' the Win32 Registry API calls.  This function closes the key that is created.
  '-----------------------------------------------------------------------------------------------
  Dim hkey As Long
  Dim lMainKey As Long
  Dim sSubKey As String
  Dim lReturnCode As Long
  Dim sClassKey As String
  Dim hNewKey As Long
  Dim lDisp As Long

  ' Parse out the main key and sub key info
  lReturnCode = ParseKeys(vsKey, lMainKey, sSubKey)
  If lReturnCode <> ErrorSuccess Then
    Reg32CreateNewKey = lReturnCode
    Exit Function
  End If

  ' Initialize the fields for the API call
  sClassKey = ""

  lReturnCode = RegCreateKeyEx(lMainKey, sSubKey, 0&, sClassKey,
RegOptionNonVolatile, KeyAllAccess, 0&, hNewKey, lDisp)
```

```
  If lReturnCode = ErrorSuccess Then
    If lDisp = RegCreatedNewKey Then
      Reg32CreateNewKey = ErrorSuccess
    Else
      Reg32CreateNewKey = ErrorKeyAlreadyExists
    End If
    ' CreateKey opens the key, so close it
    lReturnCode = RegCloseKey(hNewKey)
  Else
    Reg32CreateNewKey = lReturnCode
  End If
End Function

Public Function Reg32DeleteKey(ByVal vsKey As String) As Long
  '------------------------------------------------------------------------------------------------
  ' Deletes a key from the registry. key$ is a fully qualified Registry key with subkeys
separated by a
  ' backslash (\). This routine is operating system dependent.  If running under Win95 it
will delete the
  ' key specified and all of its child keys.  If running under WinNT it will only delete a
key if there are
  ' no child keys. Returns ErrorSuccess (0) if the key was deleted, otherwise returns
whatever error was
  ' generated by the Win32 Registry API calls.
  '------------------------------------------------------------------------------------------------
  Dim hkey As Long
  Dim lMainKey As Long
  Dim sSubKey As String
  Dim lReturnCode As Long

  ' Parse out the main key and sub key info
  lReturnCode = ParseKeys(vsKey, lMainKey, sSubKey)
  If lReturnCode <> ErrorSuccess Then
    Reg32DeleteKey = lReturnCode
    Exit Function
  End If

  ' Open the key with write access
  lReturnCode = RegOpenKeyEx(lMainKey, sSubKey, 0&, KeyWrite, hkey)
  If lReturnCode <> ErrorSuccess Then
    Reg32DeleteKey = lReturnCode
    Exit Function
  End If

  ' Delete the key--this is o/s dependant
```

```
  lReturnCode = RegDeleteKey(lMainKey, sSubKey)
  Reg32DeleteKey = lReturnCode

  ' Close the open key
  lReturnCode = RegCloseKey(hkey)
End Function

Public Function Reg32DeleteValue(ByVal vsKey As String, ByVal vsName As String)
As Long
  '-----------------------------------------------------------------------------------------------
  ' Deletes the value entry (specified by iName) from the registry key specified by key$.
key$ is a fully
  ' qualified Registry key with subkeys separated by a backslash (\). Returns ErrorSuccess
(0) if the
  ' value was deleted and ErrorFileNotFound (2) if there was no value with that name
within the specified
  ' key, otherwise returns whatever error was generated by the Win32 Registry API calls.
  '-----------------------------------------------------------------------------------------------
  Dim hkey As Long
  Dim lMainKey As Long
  Dim sSubKey As String
  Dim lReturnCode As Long

  ' Parse out the main key and sub key info
  lReturnCode = ParseKeys(vsKey, lMainKey, sSubKey)
  If lReturnCode <> ErrorSuccess Then
    Reg32DeleteValue = lReturnCode
    Exit Function
  End If

  ' Open the key with write access
  lReturnCode = RegOpenKeyEx(lMainKey, sSubKey, 0&, KeyWrite, hkey)
  If lReturnCode <> ErrorSuccess Then
    Reg32DeleteValue = lReturnCode
    Exit Function
  End If

  ' Delete the value entry
  lReturnCode = RegDeleteValue(hkey, vsName)
  Reg32DeleteValue = lReturnCode

  ' Close the open key
  lReturnCode = RegCloseKey(hkey)
End Function
```

```
Public Function Reg32GetValue(ByVal vsKey As String, ByVal vsName As String,
ByRef rlType As Long, ByRef rsValue As String) As Long
  '------------------------------------------------------------------------------------------------
  ' Retrieves a value for an entry (specified by vsName) from the registry key specified by
key$. key$ is a
  ' fully qualified Registry key with subkeys separated by a backslash (\). Returns the
value of the entry
  ' (in string format) in rsValue and its data type in rlType&. Returns ErrorSuccess (0) if
the value was
  ' found, otherwise returns whatever error was generated by the Win32 Registry API
calls.
  '------------------------------------------------------------------------------------------------
  Dim hkey As Long
  Dim lMainKey As Long
  Dim sSubKey As String
  Dim lReturnCode As Long
  Dim sItemData As String
  Dim lItemDataLen As Long
  Dim lItemDataType As Long
  Dim sItemName As String

  ' Parse out the main key and sub key info
  lReturnCode = ParseKeys(vsKey, lMainKey, sSubKey)
  If lReturnCode <> ErrorSuccess Then
    Reg32GetValue = lReturnCode
    Exit Function
  End If

  ' Open the key with read access
  lReturnCode = RegOpenKeyEx(lMainKey, sSubKey, 0&, KeyRead, hkey)
  If lReturnCode <> ErrorSuccess Then
    Reg32GetValue = lReturnCode
    Exit Function
  End If

  ' Get Key Information for this key from the registry
  kInfo.hkey = hkey
  lReturnCode = GetKeyInfo()
  If lReturnCode <> ErrorSuccess Then
    Reg32GetValue = lReturnCode
    Exit Function
  End If

  ' Initialize the fields for the API call
  sItemName = vsName
```

```
  sItemData = Space$(kInfo.MaxData + 1)
  lItemDataLen = Len(sItemData)

  lReturnCode = RegQueryValueEx(hkey, sItemName, 0&, lItemDataType, ByVal
sItemData, lItemDataLen)
  If lReturnCode = ErrorSuccess Then
    rlType = lItemDataType
    rsValue = ConvertDataToString(ByVal lItemDataType, ByVal sItemData, ByVal
lItemDataLen)
  Else
    ' Unanticipated error--get out
    Reg32GetValue = lReturnCode
  End If

  ' Close the open key
  lReturnCode = RegCloseKey(hkey)
End Function

Public Function Reg32ValueExists(ByVal vsKey As String, ByVal vsName As String)
As Boolean
  '-----------------------------------------------------------------------------------------------
  ' Determines whether a specified value entry exists in the registry (true) or not (false).
  '-----------------------------------------------------------------------------------------------
  Dim hkey As Long
  Dim lMainKey As Long
  Dim sSubKey As String
  Dim lReturnCode As Long
  Dim sItemData As String
  Dim lItemDataLen As Long
  Dim lItemDataType As Long
  Dim sItemName As String

  ' Parse out the main key and sub key info
  lReturnCode = ParseKeys(vsKey, lMainKey, sSubKey)
  If lReturnCode <> ErrorSuccess Then
    Reg32ValueExists = False
    ' Reg32GetValue = lReturnCode
    Exit Function
  End If

  ' Open the key with read access
  lReturnCode = RegOpenKeyEx(lMainKey, sSubKey, 0&, KeyRead, hkey)
  If lReturnCode <> ErrorSuccess Then
    Reg32ValueExists = False
    ' Reg32GetValue = lReturnCode
```

```
    Exit Function
  End If

  ' Get Key Information for this key from the registry
  kInfo.hkey = hkey
  lReturnCode = GetKeyInfo()
  If lReturnCode <> ErrorSuccess Then
    Reg32ValueExists = False
    ' Reg32GetValue = lReturnCode
    Exit Function
  End If

  ' Initialize the fields for the API call
  sItemName = vsName
  sItemData = Space$(kInfo.MaxData + 1)
  lItemDataLen = Len(sItemData)

  lReturnCode = RegQueryValueEx(hkey, sItemName, 0&, lItemDataType, ByVal
sItemData, lItemDataLen)
  If lReturnCode = ErrorSuccess Then
    Reg32ValueExists = True
    ' rlType = lItemDataType
    ' rsValue = ConvertDataToString(ByVal lItemDataType, ByVal sItemData, ByVal
lItemDataLen)
  Else
    Reg32ValueExists = False
    ' Reg32GetValue = lReturnCode
  End If

  ' Close the open key
  lReturnCode = RegCloseKey(hkey)
End Function

Public Function Reg32SetValue(ByVal vsKey As String, ByVal vsName As String,
ByVal vlType As Long, ByVal vsValue As String, ByVal vbCreateKey As Boolean) As
Long
  '-----------------------------------------------------------------------------------------------
  ' Sets a value for an entry (specified by vsName) in the registry key specified by
vsKey$. vsKey$ is a fully
  ' qualified Registry key with subkeys separated by a backslash (\). vsValue$ is the value
of the entry and
  ' vlType is the data type.  Currently supports data types of RegSz (strings) and
RegDWord (32-bit
  ' numbers), all other requests generate an error.  If vbCreateKey is true then the function
will attempt to
```

```
  ' create the key if it does not exist. Returns ErrorSuccess (0) if the value was written,
ErrorFileNotFound (2)
  ' if the key does not exist and vbCreateKey is false, ErrorUnsupportedType (7000) if an
unsupported data
  ' type was requested,  otherwise returns whatever error was generated by the Win32
Registry API calls.
  '-----------------------------------------------------------------------------------------------
  Dim hkey As Long
  Dim lMainKey As Long
  Dim sSubKey As String
  Dim lReturnCode As Long
  Dim sItemData As String
  Dim lItemDataType As Long
  Dim sItemName As String

  sItemData = vsValue
  lItemDataType = vlType
  sItemName = vsName

  ' Parse out the main key and sub key info
  lReturnCode = ParseKeys(vsKey, lMainKey, sSubKey)
  If lReturnCode <> ErrorSuccess Then
    Reg32SetValue = lReturnCode
    Exit Function
  End If

  ' If they want the key created, then create it first
  If vbCreateKey Then
    lReturnCode = Reg32KeyExists(ByVal vsKey)
    If lReturnCode = ErrorFileNotFound Then
      ' Doesn't exist so we have to create it
      lReturnCode = Reg32CreateNewKey(vsKey)
      If lReturnCode <> ErrorSuccess Then
      Reg32SetValue = lReturnCode
      Exit Function
    End If
    ElseIf lReturnCode <> ErrorSuccess Then
      Reg32SetValue = lReturnCode
      Exit Function
    End If
  End If

  ' Open the key with write access
  lReturnCode = RegOpenKeyEx(lMainKey, sSubKey, 0&, KeyWrite, hkey)
  If lReturnCode <> ErrorSuccess Then
```

```
    Reg32SetValue = lReturnCode
    Exit Function
  End If

  ' Write the value to the registry
  Select Case vlType
    Case RegMultiSz   ' Multiple zero-terminated strings
      Reg32SetValue = ErrorUnsupportedType
      lReturnCode = RegCloseKey(hkey)
      Exit Function
    Case RegSz   ' Zero-terminated strings
      lReturnCode = RegSetValueEx(hkey, sItemName, 0&, lItemDataType, ByVal
sItemData, CLng(Len(sItemData)))
    Case RegExpandSz   ' Zero-terminated strings with environment variable references
      Reg32SetValue = ErrorUnsupportedType
      lReturnCode = RegCloseKey(hkey)
      Exit Function
    Case RegDWord
      ' Need to make sure the value is a 32-bit number
      sItemData = Left(Trim(vsValue), 8)
      If Left$(sItemData, 2) <> "&H" Then
        sItemData = "&H" & Left(Trim(sItemData), 8)
      End If
      If Len(sItemData) <= 6 Then
        sItemData = sItemData & "&"
      End If
      lReturnCode = RegSetValueEx(hkey, sItemName, 0&, lItemDataType,
CLng(Val(sItemData)), 4&)
    Case RegBinary   ' Binary values
      Reg32SetValue = ErrorUnsupportedType
      lReturnCode = RegCloseKey(hkey)
      Exit Function
    Case Else
      Reg32SetValue = ErrorUnsupportedType
      lReturnCode = RegCloseKey(hkey)
      Exit Function
  End Select

  Reg32SetValue = lReturnCode
  ' Close the open key
  lReturnCode = RegCloseKey(hkey)
End Function

Public Function Reg32EnumValues(ByVal vsKey As String) As Long
  '-----------------------------------------------------------------------------------------------
```

```
  ' Enumerates all the value entry names in the registry for the registry key specified by
vsKey. vsKey is a
  ' fully qualified Registry key with subkeys separated by a backslash (\). The results of
the enumeration
  ' are placed in the vInfo() structure (Public variable defined in this module) which can
then be iterated
  ' through as an array for the entry names (vInfo.Item), values (vInfo.ItemValue) and
data types
  ' (vInfo.ItemType). Returns ErrorSuccess (0) if successful, otherwise returns whatever
error was
  ' generated by the Win32 Registry API calls.
  '
  ' NOTE: vInfo(0) does not contain a valid entry -- the first value from the registry is
placed in vInfo(1),
  ' the second in vInfo(2) and so on. This function returns ErrorSuccess even if there are
no entries
  ' specified in the registry for the specified key (in which case the upper and lower
bounds of vInfo will
  ' be zero.  On ErrorSuccess you should iterate through the vInfo structure from 1 to
UBound(vInfo) to
  ' retrieve all the registry entries.
  '-----------------------------------------------------------------------------------------------

  Dim hkey As Long
  Dim lMainKey As Long
  Dim sSubKey As String
  Dim sItemName As String
  Dim lItemNameLen As Long
  Dim sItemData As String
  Dim lItemDataLen As Long
  Dim lItemDataType As Long
  Dim lReturnCode As Long
  Dim lKeyIndex As Long

  ' Clear out the vInfo array
  ReDim vInfo(0 To 0)

  ' Parse out the main key and sub key info
  lReturnCode = ParseKeys(vsKey, lMainKey, sSubKey)
  If lReturnCode <> ErrorSuccess Then
    Reg32EnumValues = lReturnCode
    Exit Function
  End If

  ' Open the key with read access
```

```
  lReturnCode = RegOpenKeyEx(lMainKey, sSubKey, 0&, KeyRead, hkey)
  If lReturnCode <> ErrorSuccess Then
    Reg32EnumValues = lReturnCode
    Exit Function
  End If

  ' Get Key Information for this key from the registry
  kInfo.hkey = hkey
  lReturnCode = GetKeyInfo()
  If lReturnCode <> ErrorSuccess Then
    Reg32EnumValues = lReturnCode
    Exit Function
  End If

  ' Loop through and get all the child keys
  lKeyIndex = 0&
  Do While lReturnCode = ErrorSuccess

TryEnumValueAgain:
    ' Initialize the fields for the API call
    sItemName = Space$(kInfo.MaxValue + 1)
    lItemNameLen = CLng(Len(sItemName))
    sItemData = Space$(kInfo.MaxData + 1)
    lItemDataLen = CLng(Len(sItemData))

    lReturnCode = RegEnumValue(hkey, lKeyIndex, sItemName, lItemNameLen, 0&,
lItemDataType, ByVal sItemData, lItemDataLen)
    If lReturnCode = ErrorSuccess Then
      ReDim Preserve vInfo(LBound(vInfo) To UBound(vInfo) + 1)
      ' Add the entry to the collection
      If lItemNameLen = 0 Then
        vInfo(UBound(vInfo)).Item = "(Default)"
      Else
        vInfo(UBound(vInfo)).Item = Mid$(sItemName, 1, lItemNameLen)
      End If
      vInfo(UBound(vInfo)).ItemValue = ConvertDataToString(ByVal lItemDataType,
ByVal sItemData, ByVal lItemDataLen)
      vInfo(UBound(vInfo)).ItemType = lItemDataType
      ElseIf lReturnCode = ErrorMoreData Then
        ' Buffer too small--increment and try again
        kInfo.MaxData = kInfo.MaxData + 5
        kInfo.MaxValue = kInfo.MaxValue + 5
        GoTo TryEnumValueAgain
      ElseIf lReturnCode = ErrorNoMoreItems Then
        ' No more child keys--get out
```

```
        Reg32EnumValues = ErrorSuccess
        Exit Do
      ElseIf lReturnCode <> ErrorSuccess Then
        ' Unanticipated error--get out
        Reg32EnumValues = lReturnCode
        Exit Do
      End If
    lKeyIndex = lKeyIndex + 1
  Loop

  ' Close the open key
  lReturnCode = RegCloseKey(hkey)
End Function

Public Function Reg32EnumChildKeys(ByVal vsParentKey As String, ByRef
rcolResults As Collection) As Long
  '-------------------------------------------------------------------------------------------------
  ' Enumerates all the child keys in the registry for the registry key specified by
vsParentKey$. vsParentKey$
  ' is a fully qualified Registry key with subkeys separated by a backslash (\). The results
of the
  ' enumeration are placed into the rcolResults collection parameter passed to this
function.  Returns
  ' ErrorSuccess (0) if successful, otherwise returns whatever error was generated by the
Win32 Registry
  ' API calls.
  '-------------------------------------------------------------------------------------------------
  Dim hkey As Long
  Dim lMainKey As Long
  Dim sSubKey As String
  Dim lSubKeyLen As Long
  Dim sClassKey As String
  Dim lClassKeyLen As Long
  Dim lReturnCode As Long
  Dim lKeyIndex As Long

  ' Clear out the collection object
  For lKeyIndex = 1 To rcolResults.Count
    rcolResults.Remove 1
  Next lKeyIndex

  ' Parse out the main key and sub key info
  lReturnCode = ParseKeys(vsParentKey, lMainKey, sSubKey)
  If lReturnCode <> ErrorSuccess Then
    Reg32EnumChildKeys = lReturnCode
```

```
    Exit Function
  End If

  ' Open the key with read access
  lReturnCode = RegOpenKeyEx(lMainKey, sSubKey, 0&, KeyRead, hkey)
  If lReturnCode <> ErrorSuccess Then
    Reg32EnumChildKeys = lReturnCode
    Exit Function
  End If

  ' Get Key Information for this key from the registry
  kInfo.hkey = hkey
  lReturnCode = GetKeyInfo()
  If lReturnCode <> ErrorSuccess Then
    Reg32EnumChildKeys = lReturnCode
    Exit Function
  End If

  ' Loop through and get all the child keys
  lKeyIndex = 0&
  Do While lReturnCode = ErrorSuccess

TryAgain:
    ' Initialize the fields for the API call
    sSubKey = Space$(kInfo.MaxKey + 1)
    lSubKeyLen = CLng(Len(sSubKey))
    sClassKey = Space$(kInfo.MaxClass + 1)
    lClassKeyLen = CLng(Len(sClassKey))

    lReturnCode = RegEnumKeyEx(hkey, lKeyIndex, sSubKey, lSubKeyLen, 0&,
sClassKey, lClassKeyLen, kInfo.LastWrite)
    If lReturnCode = ErrorSuccess Then
      If InStr(sSubKey, Chr$(0)) > 1 Then
        ' Add the subkey to the collection
        sSubKey = Left$(sSubKey, InStr(sSubKey, Chr$(0)) - 1)
        rcolResults.Add sSubKey, sSubKey
      End If
      ElseIf lReturnCode = ErrorMoreData Then
        ' Buffer too small--increment and try again
        kInfo.MaxKey = kInfo.MaxKey + 5
        kInfo.MaxClass = kInfo.MaxClass + 5
        GoTo TryAgain
      ElseIf lReturnCode = ErrorNoMoreItems Then
        ' No more child keys--get out
        Reg32EnumChildKeys = ErrorSuccess
```

```
        Exit Do
      ElseIf lReturnCode <> ErrorSuccess Then
        ' Unanticipated error--get out
        Reg32EnumChildKeys = lReturnCode
        Exit Do
      End If
    lKeyIndex = lKeyIndex + 1
  Loop

  ' Close the open key
  lReturnCode = RegCloseKey(hkey)
End Function

Private Function GetMainKey(ByVal vsKeyName As String) As Long
  '-----------------------------------------------------------------------------------------------
  ' Converts the string description of a registry key to its 32-bit key handle. Used
internally by other
  ' functions in this module. Should not be called directly outside of this module.
  '-----------------------------------------------------------------------------------------------
  Select Case vsKeyName
    Case "HKEY_CLASSES_ROOT"
      GetMainKey = HkeyClassesRoot
    Case "HKEY_CURRENT_USER"
      GetMainKey = HkeyCurrentUser
    Case "HKEY_LOCAL_MACHINE"
      GetMainKey = HkeyLocalMachine
    Case "HKEY_USERS"
      GetMainKey = HkeyUsers
    Case "HKEY_PERFORMANCE_DATA"
      GetMainKey = HkeyPerformanceData
    Case "HKEY_CURRENT_CONFIG"
      GetMainKey = HkeyCurrentConfig
    Case "HKEY_DYN_DATA"
      GetMainKey = HkeyDynData
  End Select
End Function

Public Function ParseKeys(ByVal vsRegKey As String, ByRef rlSysKey As Long,
ByRef sSubKey As String) As Long
  '-----------------------------------------------------------------------------------------------
  ' Parses a registry key into the key handle for the system key (rlSysKey) and the
remainder (sSubKey).
  ' Used internally by other functions in this module. Should not be called directly outside
of this module.
  '-----------------------------------------------------------------------------------------------
```

```
  If UCase$(Left$(vsRegKey, 5)) <> "HKEY_" Then
    ParseKeys = ErrorBadKey
    Exit Function
  ElseIf InStr(vsRegKey, "\") = 0 Then
    rlSysKey = GetMainKey(vsRegKey)
    sSubKey = ""
  Else
    rlSysKey = GetMainKey(Left$(vsRegKey, InStr(vsRegKey, "\") - 1))
    sSubKey = Right$(vsRegKey, Len(vsRegKey) - InStr(vsRegKey, "\"))
  End If

  ' Make sure it is a valid key
  If rlSysKey < HkeyClassesRoot Or rlSysKey > HkeyDynData Then
    ParseKeys = ErrorBadKey
  Else
    ParseKeys = ErrorSuccess
  End If
End Function

Private Function GetKeyInfo() As Long
  '-----------------------------------------------------------------------------------------------
  ' Calls the RegQueryInfoKey API to get the info for a registry key and stores the
information in the kInfo
  ' structure
  '-----------------------------------------------------------------------------------------------
  kInfo.Class = Space$(255)
  kInfo.ClassLen = CLng(Len(kInfo.Class))
  GetKeyInfo = RegQueryInfoKey(kInfo.hkey, kInfo.Class, kInfo.ClassLen, 0&,
kInfo.KeyCnt, kInfo.MaxKey, kInfo.MaxClass, kInfo.Values, kInfo.MaxValue,
kInfo.MaxData, kInfo.Security, kInfo.LastWrite)
End Function

Private Function ConvertDataToString(ByVal vlType As Long, ByVal vsItem As String,
ByVal vlLength As Long) As String
  '-----------------------------------------------------------------------------------------------
  ' Converts the various data types from the registry to a display string
  '-----------------------------------------------------------------------------------------------
  Dim sValue As String
  Dim sTempBinary As String
  Dim lTempIndex As Long

  If Len(vsItem) = 0 Then
    sValue = ""
  Else
    sValue = Mid$(vsItem, 1, vlLength)
```

```
    Select Case vlType
      Case RegMultiSz
        ' Multiple zero-terminated strings--strip out the zeros and replace with spaces so
they all show up
        Do While InStr(sValue, Chr$(0))
          sValue = Left$(sValue, InStr(sValue, Chr$(0)) - 1) & " " & Right$(sValue,
Len(sValue) - InStr(sValue, Chr$(0)))
        Loop
      Case RegSz
        'zero-terminated strings
        sValue = Left$(sValue, vlLength - 1)
      Case RegExpandSz
        ' Zero-terminated strings that may contain environment variable references
        sValue = Left$(sValue, vlLength - 1)
      Case RegFullResourceDescriptor
        ' Can't do anything with these--they require a special editor to see them
        sValue = "RegFullResourceDescriptor"
      Case RegDWord
        ' 32-bit unsigned integers--need to manipulate values above 7FFFFFFF to appear
as positive
        fTempDbl = Asc(Mid$(sValue, 1, 1)) + &H100& * Asc(Mid$(sValue, 2, 1)) +
&H10000 * Asc(Mid$(sValue, 3, 1)) + &H1000000 * CDbl(Asc(Mid$(sValue, 4, 1)))
        If fTempDbl > &H7FFFFFFF Then
          sValue = Hex$(fTempDbl - 4294967296#)
        Else
          sValue = Hex$(fTempDbl)
        End If
      Case RegBinary
        ' Binary values
        For lTempIndex = 1 To Len(sValue)
          sTempBinary = sTempBinary & Format$(Hex(Asc(Mid$(sValue, lTempIndex,
1))), "00") & " "
        Next lTempIndex
        sValue = sTempBinary
    End Select
  End If

  ConvertDataToString = sValue
End Function

Public Function FetchRegBoolean(ByVal sRegKey As String, ByVal sRegName As
String) As Boolean
  '-----------------------------------------------------------------------------------------
  ' Fetches the item (sRegName) from the registry as a Boolean value.
  '-----------------------------------------------------------------------------------------
```

```
  Dim sValue As String
  Dim lType As Long
  Dim lReturnCode As Long
  ' Initialize the return parameter
  sValue = Space$(255)
  ' Fetch the registry entry
  lReturnCode = Reg32GetValue(sRegKey, sRegName, lType, sValue)
  ' See what we got back
  If lReturnCode = ErrorSuccess Then
    If UCase$(Trim$(CStr(sValue))) = "TRUE" Then
      FetchRegBoolean = True
    Else
      FetchRegBoolean = False
    End If
  Else
    FetchRegBoolean = False
  End If
End Function

Public Function FetchRegLong(ByVal sRegKey As String, ByVal sRegName As String)
As Long
  '-----------------------------------------------------------------------------------------------
  ' Fetches the item (sRegName) from the registry as a long value.
  '-----------------------------------------------------------------------------------------------
  Dim sValue As String
  Dim lType As Long
  Dim lReturnCode As Long
  ' Initialize the return parameter
  sValue = Space$(255)
  ' Fetch the registry entry
  lReturnCode = Reg32GetValue(sRegKey, sRegName, lType, sValue)
  ' See what we got back
  If lReturnCode = ErrorSuccess Then
    If IsNumeric(sValue) Then
      FetchRegLong = CLng(sValue)
    Else
      FetchRegLong = 0
    End If
  Else
    FetchRegLong = 0
  End If
End Function

Public Function FetchRegString(ByVal sRegKey As String, ByVal sRegName As String)
As String
```

```
  '-----------------------------------------------------------------------------------------------
  ' Fetches the item (sRegName) from the registry as a string value.
  '-----------------------------------------------------------------------------------------------
  Dim sValue As String
  Dim lType As Long
  Dim lReturnCode As Long
  ' Initialize the return parameter
  sValue = Space$(255)
  ' Fetch the registry entry
  lReturnCode = Reg32GetValue(sRegKey, sRegName, lType, sValue)
  ' See what we got back
  If lReturnCode = ErrorSuccess Then
    FetchRegString = Trim$(sValue)
  Else
    FetchRegString = ""
  End If
End Function

Public Sub PutRegBoolean(ByVal sRegKey As String, ByVal sRegName As String,
ByVal bRegValue As Boolean, ByVal bCreateKey As Boolean)
  '-----------------------------------------------------------------------------------------------
  ' Stores a boolean value in the registry as a string "True" or "False".
  '-----------------------------------------------------------------------------------------------
  Dim lReturnCode As Long

  ' Write the appropriate value to the registry
  If (bRegValue) Then
    lReturnCode = Reg32SetValue(sRegKey, sRegName, RegSz, "True", bCreateKey)
  Else
    lReturnCode = Reg32SetValue(sRegKey, sRegNamé, RegSz, "False", bCreateKey)
  End If

  ' See what we got back
  If lReturnCode <> ErrorSuccess Then
    Err.Raise vbObjectError + lReturnCode
  End If
End Sub

Public Sub PutRegLong(ByVal sRegKey As String, ByVal sRegName As String, ByVal
lRegValue As Long, ByVal bCreateKey As Boolean)
  '-----------------------------------------------------------------------------------------------
  ' Stores a numeric value in the registry as a DWord.
  '-----------------------------------------------------------------------------------------------
  Dim lReturnCode As Long
```

```
  ' Write the appropriate value to the registry
  lReturnCode = Reg32SetValue(sRegKey, sRegName, RegDWord, lRegValue,
bCreateKey)

  ' See what we got back
  If lReturnCode <> ErrorSuccess Then
    Err.Raise vbObjectError + lReturnCode
  End If
End Sub

Public Sub PutRegString(ByVal sRegKey As String, ByVal sRegName As String, ByVal
sRegValue As String, ByVal bCreateKey As Boolean)
  '-----------------------------------------------------------------------------------------------
  ' Stores a string value in the registry".
  '-----------------------------------------------------------------------------------------------
  Dim lReturnCode As Long

  ' Write the appropriate value to the registry
  If Trim$(sRegValue) = "" Then
    ' Get error 87 from the API if you try to write nothing
    lReturnCode = Reg32SetValue(sRegKey, sRegName, RegSz, Chr$(0), bCreateKey)
  Else
    lReturnCode = Reg32SetValue(sRegKey, sRegName, RegSz, Trim$(sRegValue),
bCreateKey)
  End If

  ' See what we got back
  If lReturnCode <> ErrorSuccess Then
    Err.Raise vbObjectError + lReturnCode
  End If
End Sub
```

Module 2 About form Functions and Subroutines

```
'------------------------------------------------------------------------------------------------
' Force explicit declaration of all variables in that module
'------------------------------------------------------------------------------------------------
Option Explicit

Private Sub cmdOK_Click()
  '------------------------------------------------------------------------------------------------
  ' Unload the About form
  '------------------------------------------------------------------------------------------------
  Unload Me
End Sub

Private Sub CmdSysInfo_Click()
  '------------------------------------------------------------------------------------------------
  ' Display system information
  '------------------------------------------------------------------------------------------------
  On Error GoTo CmdSysInfoError

  Dim lReturnCode As Long
  Dim sProgramPath As String

  ' find the path of the system info program
  sProgramPath =
FetchRegString("HKEY_LOCAL_MACHINE\Software\Microsoft\Shared
Tools\MSInfo", "Path")

  ' put up a message if the program's not available
  If sProgramPath = "" Then
    MsgBox "System Information not available.", vbInformation + vbOKOnly, "System
Information"
    Exit Sub
  End If

  ' run Microsoft Info program
  lReturnCode = Shell(sProgramPath, 1)

  Exit Sub

CmdSysInfoError:
  Err.Raise Err.Number, Err.Description
  Exit Sub
End Sub
```

```
Private Sub Form_Load()
  '-------------------------------------------------------------------------------------------------
  ' Form load
  '-------------------------------------------------------------------------------------------------
  On Error GoTo FormLoadError
  CenterForm Me

  'CUSTOM => modify application's full title
  lblTitle.Caption = "Blind Phone System"
  'CUSTOM => modify version information
  lblVersion.Caption = "Version 0.10"
  Me.Caption = "About" & Space(1) & lblTitle.Caption
  Dim txt$
  txt$ = "Warning: This computer program is protected by copyright law and
international treaties. "
  txt$ = txt$ & "Unauthorized reproduction or distribution of this program, "
  txt$ = txt$ & "or any portion of it, may result in severe civil and criminal penalties, "
  txt$ = txt$ & "and will be prosecuted to the maximum extent possible under law."
  lblWarning.Caption = txt$

  Exit Sub

FormLoadError:
  Err.Raise Err.Number, Err.Description
  Exit Sub
End Sub

Private Sub Form_Unload(Cancel As Integer)
  '-------------------------------------------------------------------------------------------------
  ' Release the memory and system resources associated with the About form
  '-------------------------------------------------------------------------------------------------
  On Error GoTo FormUnloadError
  Set fAbout = Nothing

  Exit Sub

FormUnloadError:
  Err.Raise Err.Number, Err.Description
  Exit Sub
End Sub
```

Module 3 Phone.Bas

```
'-----------------------------------------------------------------------------------------------------
' Force explicit declaration of all variables in that module
'-----------------------------------------------------------------------------------------------------
Option Explicit
'-----------------------------------------------------------------------------------------------------
' Definition of symbols used to define the variable's datatype
' b : Boolean variable
' c : Constant
' DB : Database
' i : Integer variable
' l : Long variable
' p : Public variable
' Rs : RecordSet
' s : String variable
'-----------------------------------------------------------------------------------------------------
Public Const pclSuccess As Long = 0
Public pDBase As Database
Public pQueryDef As QueryDef
Public pRsName As Recordset
Public pRsLocation As Recordset
Public pRsPhone As Recordset

Public psLocationID As String
Public psNameID As String
Public plNameID As Long
Public plLocationID As Long
Public plPhoneID As Long
Public piAction As Integer  ' Control Action within save Button
Public pbBlindUser As Boolean

Public piResponse As Integer
Public plReturnValue As Long

Public poVoiceText As Object
Public piSpeechType As Integer
Public plSpeechPriority As Long
Public plSpeechSpeed As Long
Public pbRegisterFirstTime As Boolean

' If Name has focus then piEditAddDelete = 1 (Edit, Add, Or Delete a name)
' If Location has focus then piEditAddDelete = 2 (Edit, Add, Or Delete a location)
```

```
' If Phone display has focus then piEditAddDelete = 3 (Edit, Add, Or Delete a phone
number)
' Else piEditAddDelete = 0
Public piEditAddDelete As Integer

' The  Telephony  API  is jointly copyrighted by Intel and Microsoft.  You are granted  a
royalty
' free worldwide, unlimited license to make copies, and use the API/SPI for making
' applications/drivers that interface with the specification provided that this paragraph and
the
' Intel/Microsoft copyright statement is maintained as is in the text and source code files.
'
' Copyright 1992, 1993 Intel/Microsoft, all rights reserved.
'
'=============================================================================
'-----------------------------------------------------------------------------
' Simple Telephony Constants.
'-----------------------------------------------------------------------------
Public Const pclTAPI_REPLY = &H400& + 99&

Public Const pclTAPIERR_CONNECTED = 0&
Public Const pclTAPIERR_DROPPED = -1&
Public Const pclTAPIERR_NOREQUESTRECIPIENT = -2&
Public Const pclTAPIERR_REQUESTQUEUEFULL = -3&
Public Const pclTAPIERR_INVALDESTADDRESS = -4&
Public Const pclTAPIERR_INVALWINDOWHANDLE = -5&
Public Const pclTAPIERR_INVALDEVICECLASS = -6&
Public Const pclTAPIERR_INVALDEVICEID = -7&
Public Const pclTAPIERR_DEVICECLASSUNAVAIL = -8&
Public Const pclTAPIERR_DEVICEIDUNAVAIL = -9&
Public Const pclTAPIERR_DEVICEINUSE = -10&
Public Const pclTAPIERR_DESTBUSY = -11&
Public Const pclTAPIERR_DESTNOANSWER = -12&
Public Const pclTAPIERR_DESTUNAVAIL = -13&
Public Const pclTAPIERR_UNKNOWNWINHANDLE = -14&
Public Const pclTAPIERR_UNKNOWNREQUESTID = -15&
Public Const pclTAPIERR_REQUESTFAILED = -16&
Public Const pclTAPIERR_REQUESTCANCELLED = -17&
Public Const pclTAPIERR_INVALPOINTER = -18&

Public Const pclTAPIMAXDESTADDRESSSIZE = 80&
Public Const pclTAPIMAXAPPNAMESIZE = 40&
Public Const pclTAPIMAXCALLEDPARTYSIZE = 40&
Public Const pclTAPIMAXCOMMENTSIZE = 80&
Public Const pclTAPIMAXDEVICECLASSSIZE = 40&
```

```
Public Const pclTAPIMAXDEVICEIDSIZE = 40&

'=============================================================================
'-----------------------------------------------------------------------------
' Simple Telephony prototypes
'-----------------------------------------------------------------------------
' Parameter list:
' 1. lpszDestAddress: Specifies a pointer to a memory location where the NULL-
terminated
' destination address of the call request is located.
' 2. lpszAppName: Specifies a pointer to a memory location where the ASCII NULL-
terminated
' user-friendly application name of the call request is located.
' 3. lpszCalledParty: Specifies a pointer to a memory location where the ASCII NULL-
terminated
' called party name for the called party of the call is located.
' 4. lpszComment: Specifies a pointer to a memory location where the ASCII NULL-
terminated
' comment about the call is located.
Declare Function tapiRequestMakeCall Lib "TAPI32.DLL" (ByVal lpszDestAddress As
String, ByVal lpszAppName As String, ByVal lpszCalledParty As String, ByVal
lpszComment As String) As Long
'-----------------------------------------------------------------------------
' Used to Close the Microsoft Phone
'-----------------------------------------------------------------------------
Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName
As String, ByVal lpWindowName As String) As Long
Declare Function PostMessage Lib "user32" Alias "PostMessageA" (ByVal hwnd As
Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
Declare Function SetActiveWindow Lib "user32" (ByVal hwnd As Long) As Long
'-----------------------------------------------------------------------------
' Retrieves the state of the specified Keyboard virtual key at the time the function is
called.
'-----------------------------------------------------------------------------
Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer

Function Alert(ByVal sMessageText As String, ByVal sMessageTitle As String, ByVal
iButtons As Integer, ByVal bHandiCap As Boolean, ByVal bSelectText As Boolean) As
Integer
  '-----------------------------------------------------------------------------
  ' Function Alert is used to display a message or to add the text to a RichTextBox.
  '-----------------------------------------------------------------------------
  On Error GoTo AlertError

  If (bHandiCap) Then
```

```
    Beep
    SpeakText sMessageTitle & ", " & sMessageText
  Else
    If ((sMessageTitle = "Exit") Or (sMessageTitle = "Confirmation")) Then
      Alert = MsgBox(sMessageText, iButtons, sMessageTitle)
    Else
      MsgBox sMessageText, iButtons, sMessageTitle
    End If
  End If

  Exit Function

AlertError:
  ErrMsg ("Alert")
  Exit Function
End Function

Public Sub CenterForm(fForm As Form)
  '-----------------------------------------------------------------------------------------------
  ' Centers a form on the screen.
  '-----------------------------------------------------------------------------------------------
  On Error GoTo CenterFormError

  Dim iLeft As Integer
  Dim iTop As Integer

  ' Get left offset
  iLeft = (Screen.Width - fForm.Width) / 2

  ' Get top offset
  iTop = (Screen.Height - fForm.Height) / 2

  ' Position the form
  fForm.Move iLeft, iTop

  Exit Sub

CenterFormError:
  ErrMsg ("CenterForm")
  Exit Sub
End Sub

Public Function GetDataBase()
  '-----------------------------------------------------------------------------------------------
  ' Return the location of the Database.
```

```
  '-------------------------------------------------------------------------------------------------
  On Error GoTo GetDataBaseError

  If (Len(Trim(Command)) = 0) Then
    GetDataBase = IIf(Right(App.Path, 1) = "\", App.Path, App.Path + "\") +
"PhonBook.mdb"
  Else
    GetDataBase = Trim(Command)
  End If

  Exit Function

GetDataBaseError:
  ErrMsg ("GetDataBase")
  Exit Function
End Function

Public Sub AutoSelect(SelObject As Control)
  '-------------------------------------------------------------------------------------------------
  ' The AutoSelect routine selects the control's entire contents
  '-------------------------------------------------------------------------------------------------
  On Error GoTo AutoSelectError

  SelObject.SelStart = 0
  If TypeOf SelObject Is MaskEdBox Then
    SelObject.SelLength = Len(SelObject.FormattedText)
  Else
    If TypeOf SelObject Is TextBox Then
      SelObject.SelLength = Len(SelObject.Text)
    End If
  End If

  Exit Sub

AutoSelectError:
  ErrMsg ("AutoSelect")
  Exit Sub
End Sub

Sub ErrMsg(sMessage As String)
  '-------------------------------------------------------------------------------------------------
  ' Error Messages.
  '-------------------------------------------------------------------------------------------------
  Dim sErrorLogFilename As String
```

```
  If (pbBlindUser) Then
    sErrorLogFilename = IIf(Right(App.Path, 1) = "\", App.Path, App.Path + "\") +
"PhoneErrorLog.txt"
    ' Create filename.
    Open sErrorLogFilename For Output As #1
    ' Outputs the current date and time according to the setting of your computer's system
date and time.
    Write #1, Now
    ' Output run time error
    Write #1, sMessage + " - " + Error + " (" & Err & ")", vbCritical, "Error"
    ' Close file.
    Close #1
    Beep
    SpeakText "Please contact customer support the system generated a run time error."
  Else
    MsgBox sMessage + " - " + Error + " (" & Err & ")", vbCritical, "Error"
    Err.Clear
  End If

  Dim lhWnd As Long

  ' Close MS Phone if active
  lhWnd = FindWindow(vbNullString, "Phone")
  If (lhWnd <> 0) Then
    plReturnValue = SetActiveWindow(lhWnd)
    plReturnValue = PostMessage(lhWnd, &H10, 0, 0&)
    lhWnd = FindWindow(vbNullString, "Phone")
    ' Select yes to Exit MS Phone
    If (lhWnd <> 0) Then
      plReturnValue = SetActiveWindow(lhWnd)
      SendKeys "{ENTER}", True
    End If
  End If
  Set fPhone = Nothing
  Set poVoiceText = Nothing
  ' Terminates execution
  End
End Sub

Public Sub RegisterVoiceText()
  '---------------------------------------------------------------------------------------------
  ' Registers an application with the Voice-Text object.
  '---------------------------------------------------------------------------------------------
   On Error Resume Next
```

```
    Set poVoiceText = Nothing
    Set poVoiceText = CreateObject("Speech.VoiceText")

    Call poVoiceText.Register("", "Blind Phone")
    Call VoiceTextErrorCheck(pciVTXT_REGISTER)
    ' Accepts the Raise and Clear methods for generating and clearing run-time errors.
    Err.Clear

    ' Enable notification
    poVoiceText.Callback = "Telephony.VoiceTextSink"
    Call VoiceTextErrorCheck(pciVTXT_CALLBACK)
   ' Accepts the Raise and Clear methods for generating and clearing run-time errors.
    Err.Clear

    plSpeechSpeed = poVoiceText.Speed

    If pbRegisterFirstTime = True Then
       ' Enable or disable necessary controls
       pbRegisterFirstTime = False
    End If
End Sub

Public Sub SpeakText(sTextToSpeech As String)
  '---------------------------------------------------------------------------------------------------
  ' Convert words in a character string into speech played over the computer speakers.
  '---------------------------------------------------------------------------------------------------
  ' plSpeechPriority = &H80
  plSpeechPriority = vtxtsp_VERYHIGH

  ' piSpeechType = &H1
  piSpeechType = vtxtst_STATEMENT

  On Error Resume Next
  DoEvents
    Call poVoiceText.Speak(sTextToSpeech, piSpeechType Or plSpeechPriority)
  ' Loop while text is being converted to speech
  Do While (poVoiceText.IsSpeaking) And (fPhone.StatusBar.Panels(1).Text =
"Speaking Done notification")
  Loop
  Call VoiceTextErrorCheck(pciVTXT_SPEAK)
  ' Accepts the Raise and Clear methods for generating and clearing run-time errors.
  Err.Clear
End Sub
```

Module 4 Phone form Functions and Subroutines

```
' Force explicit declaration of all variables in that module
'--------------------------------------------------------------------------------------------------------
Option Explicit

Private Sub AddBtn_Click()
  '--------------------------------------------------------------------------------------------------------
  ' Add a Name, Location, or a Phone Number.
  '--------------------------------------------------------------------------------------------------------
  Dim iNewPhoneID As Integer

  On Error GoTo AddBtnClickError
  Select Case piEditAddDelete
    Case 1     ' Add Name
      If (Len(Trim(NameDBCombo.Text)) > 0) Then
        If (Not (NameDBCombo.MatchedWithList)) Then
          Set pRsName = pDBase.OpenRecordset("Get New Name ID")
          Set pQueryDef = pDBase.QueryDefs("Add New Name")
          pQueryDef.Parameters("@NameID") = pRsName("NameID")
          pQueryDef.Parameters("@Name") = Trim(NameDBCombo.Text)
          pQueryDef.Execute dbFailOnError
          pQueryDef.Close
          Set pQueryDef = Nothing
          piResponse = Alert("(" & NameDBCombo.Text & ") was added Successfully.",
"Information", vbOKOnly + vbInformation, pbBlindUser, False)
          NameData.Refresh
          NameDBCombo.BoundText = Val(pRsName("NameID"))
          psNameID = NameDBCombo.BoundText
          pRsName.Close
          Set pRsName = Nothing
        Else
          piResponse = Alert("Duplicate name.", "Error", vbOKOnly + vbCritical,
pbBlindUser, False)
          NameDBCombo.Text = ""
          NameDBCombo.SetFocus
        End If
      Else
        piResponse = Alert("Please Enter the person's name to add.", "Error", vbOKOnly
+ vbCritical, pbBlindUser, False)
        NameDBCombo.Text = ""
        NameDBCombo.SetFocus
      End If
    Case 2     ' Add Location
```

```
      If (Len(Trim(LocationDBCombo.Text)) > 0) Then
        If (Not (LocationDBCombo.MatchedWithList)) Then
          Set pRsLocation = pDBase.OpenRecordset("Get New Location ID")
          Set pQueryDef = pDBase.QueryDefs("Add New Location")
          pQueryDef.Parameters("@LocationID") = pRsLocation("LocationID")
          pQueryDef.Parameters("@Location") = Trim(LocationDBCombo.Text)
          pQueryDef.Execute dbFailOnError
          pQueryDef.Close
          Set pQueryDef = Nothing
          piResponse = Alert("(" & LocationDBCombo.Text & ") was added
Successfully.", "Information", vbOKOnly + vbInformation, pbBlindUser, False)
          LocationData.Refresh
          LocationDBCombo.BoundText = Val(pRsLocation("LocationID"))
          psLocationID = LocationDBCombo.BoundText
          pRsLocation.Close
          Set pRsLocation = Nothing
        Else
          piResponse = Alert("Duplicate location.", "Error", vbOKOnly + vbCritical,
pbBlindUser, False)
          LocationDBCombo.Text = ""
          LocationDBCombo.SetFocus
        End If
      Else
        piResponse = Alert("Please Enter phone's physical location to add.", "Error",
vbOKOnly + vbCritical, pbBlindUser, False)
        LocationDBCombo.Text = ""
        LocationDBCombo.SetFocus
      End If
    Case 3     ' Add Phone
      If ((Val(psNameID) > 0) And (NameDBCombo.MatchedWithList)) Then
        If ((Val(psLocationID) > 0) And (LocationDBCombo.MatchedWithList)) Then
          If (Len(Trim(Display.Text)) > 0) Then
            ' Retrieve the phone number
            Set pQueryDef = pDBase.QueryDefs("Get Phone ID And Number")
            pQueryDef.Parameters("@NameID") = Val(psNameID)
            pQueryDef.Parameters("@LocationID") = Val(psLocationID)
            Set pRsPhone = pQueryDef.OpenRecordset
            pQueryDef.Close
            Set pQueryDef = Nothing
            If (pRsPhone.EOF) Then
              Set pRsPhone = pDBase.OpenRecordset("Get New Phone ID")
              iNewPhoneID = pRsPhone("PhoneID")
              Set pQueryDef = pDBase.QueryDefs("Add New Phone")
              pQueryDef.Parameters("@PhoneID") = iNewPhoneID
              pQueryDef.Parameters("@NameID") = Val(psNameID)
```

```
                pQueryDef.Parameters("@LocationID") = Val(psLocationID)
                pQueryDef.Parameters("@PhoneNumber") = Trim(Display.Text)
                pQueryDef.Execute dbFailOnError
                pQueryDef.Close
                Set pQueryDef = Nothing
                piResponse = Alert(NameDBCombo.Text & "'s " &
LocationDBCombo.Text & " phone number " & Display.Text & " was added
Successfully.", "Information", vbOKOnly + vbInformation, pbBlindUser, True)
                piEditAddDelete = 0
              Else
                piResponse = Alert(NameDBCombo.Text & "'s " &
LocationDBCombo.Text & " phone number already exist, Please use update to modify
it.", "Error", vbOKOnly + vbCritical, pbBlindUser, True)
              End If
              pRsPhone.Close
              Set pRsPhone = Nothing
            Else
              piResponse = Alert("Please enter " & NameDBCombo.Text & "'s " &
LocationDBCombo.Text & " phone number to be added.", "Error", vbOKOnly +
vbCritical, pbBlindUser, True)
              Display.SetFocus
            End If
          Else
            piResponse = Alert("Please Select a Phone Location.", "Error", vbOKOnly +
vbCritical, pbBlindUser, True)
            LocationDBCombo.SetFocus
          End If
        Else
          piResponse = Alert("Please Select or Add the person's name for whom you want
to add a phone number.", "Error", vbOKOnly + vbCritical, pbBlindUser, True)
          NameDBCombo.SetFocus
        End If
      Case Else
        piResponse = Alert("To add a Name, Location, or Phone number, please set focus
on the Name or Location ComboBoxs or on the Phone Display EditBox.", "Information",
vbOKOnly + vbInformation, pbBlindUser, True)
    End Select

    Exit Sub

AddBtnClickError:
    ErrMsg ("AddBtnClick")
    Exit Sub
End Sub
```

```
Private Sub AddBtn_GotFocus()
  '-------------------------------------------------------------------------------------------------
  ' Executes when the Dial Command Button object receives the focus by using the Tab
key.
  '-------------------------------------------------------------------------------------------------
  On Error GoTo AddBtnGotFocus

    If ((pbBlindUser) And (GetAsyncKeyState(&H9) = 0 Or GetAsyncKeyState(&H9) =
-32767 Or GetAsyncKeyState(&H9) = 1)) Then
    SpeakText AddBtn.Caption
  End If

  Exit Sub

AddBtnGotFocus:
  ErrMsg ("AddBtnGotFocus")
  Exit Sub
End Sub

Private Sub ClearBtn_Click()
  '-------------------------------------------------------------------------------------------------
  ' Clear phone display.
  '-------------------------------------------------------------------------------------------------
  On Error GoTo ClearBtnClickError
  Display.Text = ""

  Exit Sub

ClearBtnClickError:
  ErrMsg ("ClearBtn Click")
  Exit Sub
End Sub

Private Sub ClearBtn_GotFocus()
  '-------------------------------------------------------------------------------------------------
  ' Executes when the Clear Command Button object receives the focus.
  '-------------------------------------------------------------------------------------------------
  On Error GoTo ClearBtnGotFocusError

    If ((pbBlindUser) And (GetAsyncKeyState(&H9) = 0 Or GetAsyncKeyState(&H9) =
-32767 Or GetAsyncKeyState(&H9) = 1)) Then
    SpeakText "Clear phone display"
  End If

  Exit Sub
```

```
ClearBtnGotFocusError:
  ErrMsg ("ClearBtn GotFocus")
  Exit Sub
End Sub

Private Sub DeleteBtn_GotFocus()
  '---------------------------------------------------------------------------------------------
  ' Executes when the Dial Command Button object receives the focus by using the Tab
key.
  '---------------------------------------------------------------------------------------------
  On Error GoTo DeleteBtnGotFocus

    If ((pbBlindUser) And (GetAsyncKeyState(&H9) = 0 Or GetAsyncKeyState(&H9) =
-32767 Or GetAsyncKeyState(&H9) = 1)) Then
    SpeakText DeleteBtn.Caption
  End If

  Exit Sub

DeleteBtnGotFocus:
  ErrMsg ("DeleteBtnGotFocus")
  Exit Sub
End Sub

Private Sub DialBtn_GotFocus()
  '---------------------------------------------------------------------------------------------
  ' Executes when the Dial Command Button object receives the focus by using the Tab
key.
  '---------------------------------------------------------------------------------------------
  On Error GoTo DialBtnGotFocus

    If ((pbBlindUser) And (GetAsyncKeyState(&H9) = 0 Or GetAsyncKeyState(&H9) =
-32767 Or GetAsyncKeyState(&H9) = 1)) Then
    SpeakText DialBtn.Caption
  End If

  Exit Sub

DialBtnGotFocus:
  ErrMsg ("DialBtnGotFocus")
  Exit Sub
End Sub

Private Sub GetPhoneBtn_Click()
```

```
'-------------------------------------------------------------------------------------------------
' Retrieve the Phone number from the database.
'-------------------------------------------------------------------------------------------------
On Error GoTo GetPhoneBtnClickError
' Get Phone Number
psNameID = NameDBCombo.BoundText
psLocationID = LocationDBCombo.BoundText
If ((Val(psNameID) > 0) And (NameDBCombo.MatchedWithList)) Then
  If ((Val(psLocationID) > 0) And (LocationDBCombo.MatchedWithList)) Then
    ' Retrieve the phone number and ID
    Set pQueryDef = pDBase.QueryDefs("Get Phone ID And Number")
    pQueryDef.Parameters("@NameID") = Val(psNameID)
    pQueryDef.Parameters("@LocationID") = Val(psLocationID)
    Set pRsPhone = pQueryDef.OpenRecordset
    pQueryDef.Close
    Set pQueryDef = Nothing
    If (Not pRsPhone.EOF) Then
      Display.Text = pRsPhone("PhoneNum")
      If (pbBlindUser) Then
        piResponse = Alert(NameDBCombo.Text & "'s " & LocationDBCombo.Text &
" phone number " & Display.Text & " was retrieved.", "Information", vbOKOnly +
vbCritical, pbBlindUser, True)
      End If
    Else
      Display.Text = ""
      piResponse = Alert(NameDBCombo.Text & "'s " & LocationDBCombo.Text & "
phone number does not exist.", "Information", vbOKOnly + vbInformation, pbBlindUser,
True)
      ' Display.SetFocus
    End If
    pRsPhone.Close
    Set pRsPhone = Nothing
  Else
    piResponse = Alert("Please select a phone physical location.", "Error", vbOKOnly
+ vbCritical, pbBlindUser, True)
    LocationDBCombo.SetFocus
  End If
Else
  piResponse = Alert("Please select the name of the party you want to call.", "Error",
vbOKOnly + vbCritical, pbBlindUser, True)
  NameDBCombo.SetFocus
End If

Exit Sub
```

```
GetPhoneBtnClickError:
  ErrMsg ("GetPhoneBtnClick")
  Exit Sub
End Sub

Private Sub DeleteBtn_Click()
  '-----------------------------------------------------------------------------------------------------
  ' Delete a Name, Location, or a Phone Number
  '-----------------------------------------------------------------------------------------------------
  On Error GoTo DeleteBtnClickError
  Select Case piEditAddDelete
    Case 1   ' Delete Name
      If ((Val(psNameID) > 0) And (NameDBCombo.MatchedWithList)) Then
        'If (pbBlindUser) Then
        '  SpeakText "Delete the person's name and phone numbers?  Say yes or no."
        'End If
        Set pQueryDef = pDBase.QueryDefs("Delete Name")
        pQueryDef.Parameters("@NameID") = psNameID
        pQueryDef.Execute dbFailOnError
        pQueryDef.Close
        Set pQueryDef = Nothing
        piResponse = Alert(NameDBCombo.Text & "s' name and phone number(s) were
deleted.", "Information", vbOKOnly + vbInformation, pbBlindUser, True)
        piEditAddDelete = 1
        Display.Text = ""
        NameData.Refresh
      Else
        piResponse = Alert("Please select the person's name to deleted.", "Error",
vbOKOnly + vbCritical, pbBlindUser, True)
        NameDBCombo.SetFocus
      End If
    Case 2   ' Delete Location
      If ((Val(psLocationID) > 0) And (LocationDBCombo.MatchedWithList)) Then
        'If (pbBlindUser) Then
        '  SpeakText "Delete the selected location and the phone numbers associated with
it?  Say yes or no."
        'End If
        Set pQueryDef = pDBase.QueryDefs("Delete Location")
        pQueryDef.Parameters("@LocationID") = psLocationID
        pQueryDef.Execute dbFailOnError
        pQueryDef.Close
        Set pQueryDef = Nothing
        piResponse = Alert("The " & LocationDBCombo.Text & " location and phone
number(s) associated with it were deleted.", "Information", vbOKOnly + vbInformation,
pbBlindUser, True)
```

```
        piEditAddDelete = 2
        Display.Text = ""
        LocationData.Refresh
      Else
        piResponse = Alert("Please select a location to delete.", "Error", vbOKOnly +
vbCritical, pbBlindUser, True)
        LocationDBCombo.SetFocus
      End If
    Case 3     ' Delete Phone
      If ((Val(psNameID) > 0) And (NameDBCombo.MatchedWithList)) Then
        If ((Val(psLocationID) > 0) And (LocationDBCombo.MatchedWithList)) Then
          ' Retrieve the phone number and ID
          Set pQueryDef = pDBase.QueryDefs("Get Phone ID And Number")
          pQueryDef.Parameters("@NameID") = Val(psNameID)
          pQueryDef.Parameters("@LocationID") = Val(psLocationID)
          Set pRsPhone = pQueryDef.OpenRecordset
          pQueryDef.Close
          Set pQueryDef = Nothing
          If (Not pRsPhone.EOF) Then
            Set pQueryDef = pDBase.QueryDefs("Delete Phone")
            pQueryDef.Parameters("@PhoneID") = Val(pRsPhone("PhoneID"))
            pQueryDef.Execute dbFailOnError
            pQueryDef.Close
            Set pQueryDef = Nothing
            piResponse = Alert(NameDBCombo.Text & "'s " & LocationDBCombo.Text
& " phone number was deleted.", "Information", vbOKOnly + vbInformation,
pbBlindUser, True)
            piEditAddDelete = 0
            Display.Text = ""
            pRsPhone.Close
            Set pRsPhone = Nothing
          Else
            piResponse = Alert(NameDBCombo.Text & "'s " & LocationDBCombo.Text
& " phone number does not exist.", "Error", vbOKOnly + vbCritical, pbBlindUser, True)
            ' GetPhoneBtn.SetFocus
          End If
        Else
          piResponse = Alert("Please select the phone's physical location to delete the
phone number.", "Error", vbOKOnly + vbCritical, pbBlindUser, True)
          LocationDBCombo.SetFocus
        End If
      Else
        piResponse = Alert("Please select the person's name to delete the phone number.",
"Error", vbOKOnly + vbCritical, pbBlindUser, True)
        NameDBCombo.SetFocus
```

```
      End If
    Case Else
      piResponse = Alert("To delete a Name, Location, or Phone number, please set focus
on the Name or Location ComboBoxs or on the Phone Display EditBox.", "Information",
vbOKOnly + vbInformation, pbBlindUser, True)
  End Select

  Exit Sub

DeleteBtnClickError:
  ErrMsg ("DeleteBtnClick")
  Exit Sub
End Sub

Private Sub DialBtn_Click()
  '-------------------------------------------------------------------------------------------------
  ' Pass the phone number to the tapiRequestMakeCall TAPI32.DLL function
  ' which will activate Microsoft Phone
  '-------------------------------------------------------------------------------------------------
  On Error GoTo DialBtnClickError
  Dim sPhone_Number As String
  Dim sInfoString As String

  sPhone_Number = Trim(Display.Text)

  If (Len(sPhone_Number) = 0) Then
    piResponse = Alert("Invalid Phone number.", "Error", vbOKOnly + vbCritical,
pbBlindUser, True)
    Exit Sub
  End If
  ' Requests the establishment of a voice call. A Microsoft call-manager application is
responsible for
  ' establishing the call on behalf of the requesting application, which is then controlled
by the user's
  ' call-manager application.
  plReturnValue = tapiRequestMakeCall(sPhone_Number, "", sPhone_Number, "")
  sInfoString = "Unable to dial " & sPhone_Number & " because "
  Select Case plReturnValue
    Case pclTAPIERR_NOREQUESTRECIPIENT
      sInfoString = sInfoString & "a call-manager is unavailable."
    Case pclTAPIERR_REQUESTQUEUEFULL
      sInfoString = sInfoString & "the Windows Telephony dialing queue is full."
    Case pclTAPIERR_INVALDESTADDRESS
      sInfoString = sInfoString & "the phone number is invalid."
    Case pclTAPIERR_INVALPOINTER
```

```
      sInfoString = sInfoString & "the pointer does not reference a valid memory
location."
    Case Else
      sInfoString = sInfoString & "of an unknown problem."
  End Select
  If (plReturnValue <> pclSuccess) Then
    piResponse = Alert(sInfoString, "Error", vbOKOnly + vbCritical, pbBlindUser, True)
  End If

  Exit Sub

DialBtnClickError:
  ErrMsg ("DialBtnClick")
  Exit Sub
End Sub

Private Sub Display_GotFocus()
  '-------------------------------------------------------------------------------------------------------
  ' Executes when the Display EditBox Button object receives the focus.
  '-------------------------------------------------------------------------------------------------------
  On Error GoTo DisplayGotFocusError
  ' Edit, Add, or Delete Phone
  piEditAddDelete = 3
  If (pbBlindUser) Then
    If (Len(Trim(Display.Text)) = 0) Then
      SpeakText "Phone display"
    Else
      SpeakText Trim(Display.Text)
    End If
  End If
  AutoSelect Display

  Exit Sub

DisplayGotFocusError:
  ErrMsg ("Display GotFocus")
  Exit Sub
End Sub

Private Sub Display_KeyPress(KeyAscii As Integer)
  '-------------------------------------------------------------------------------------------------------
  ' Limit Entry to numbers or Backspace only
  '-------------------------------------------------------------------------------------------------------
  Select Case KeyAscii
```

```
    Case vbKeyBack, vbKeyEnd, vbKeyHome, vbKeyLeft, vbKeyRight, vbKeyInsert,
vbKeyDelete, 48 To 57
    Case Else
      KeyAscii = 0
  End Select
End Sub

Private Sub Form_Load()
  '-------------------------------------------------------------------------------------------------
  ' Execute when the form is loaded.
  '-------------------------------------------------------------------------------------------------
  On Error GoTo FormLoadError

  Dim sDBaseName As String

  ' Find the path of the system info program
  pbBlindUser =
FetchRegBoolean("HKEY_LOCAL_MACHINE\Software\BlindPhone", "BlindUser")

  piEditAddDelete = 0

  ' Centers the form on the screen.
  CenterForm Me

  ' Initialize Voice Text
  pbRegisterFirstTime = True

  ' Set the speed at which speech is spoken (Words per Minute)
  ' Mininum Speech Speed = 50
  ' Maximum Speech Speed  = 250
  plSpeechSpeed = 75

  ' Registers an application with the Voice-Text object.
  RegisterVoiceText

  If (pbBlindUser) Then
    mnuOptionsBlindUser(0).Checked = True
    ' Enable Voice Text
    poVoiceText.Enabled = True
    Call VoiceTextErrorCheck(pciVTXT_ATTRIB_ENABLE)
    SpeakText "Blind Phone is Ready"
  Else
    mnuOptionsBlindUser(0).Checked = False
    ' Disable Voice Text
    poVoiceText.Enabled = False
```

```
    Call VoiceTextErrorCheck(pciVTXT_ATTRIB_DISABLE)
  End If

  ' Get the DataBase from the Application Directory
  sDBaseName = GetDataBase
  NameData.DatabaseName = sDBaseName
  LocationData.DatabaseName = sDBaseName
  ' Returns a pointer to the current database
  Set pDBase = DBEngine.Workspaces(0).OpenDatabase(sDBaseName)

 ' Run MS Voice.
 'plReturnValue = Shell("C:\Program Files\Microsoft Phone\Msvoice.exe")

  Exit Sub

FormLoadError:
  ErrMsg ("Form Load")
  Exit Sub
End Sub

Private Sub Form_Unload(Cancel As Integer)
  '-------------------------------------------------------------------------------------------------
  ' Release the memory and system resources associated with the Phone form.
  '-------------------------------------------------------------------------------------------------
  On Error GoTo FormUnloadError

  Unload fPhone

  ' Exit the phone project
  Set fPhone = Nothing
  Set poVoiceText = Nothing
  End

FormUnloadError:
  ErrMsg ("Form Unload")
  Exit Sub
End Sub

Private Sub GetPhoneBtn_GotFocus()
  '-------------------------------------------------------------------------------------------------
  ' Executes when the Dial Command Button object receives the focus by using the Tab
key.
  '-------------------------------------------------------------------------------------------------
  On Error GoTo GetPhoneBtnGotFocus
```

```
    If ((pbBlindUser) And (GetAsyncKeyState(&H9) = 0 Or GetAsyncKeyState(&H9) =
-32767 Or GetAsyncKeyState(&H9) = 1)) Then
      SpeakText GetPhoneBtn.Caption
    End If

    Exit Sub

GetPhoneBtnGotFocus:
    ErrMsg ("GetPhoneBtnGotFocus")
    Exit Sub

End Sub

Private Sub LocationDBCombo_Click(Area As Integer)
    '---------------------------------------------------------------------------------------------------
    ' Execute when the user Clicks on the LocationDBCombo.
    '---------------------------------------------------------------------------------------------------
    ' Phone physical location ID
    psLocationID = LocationDBCombo.BoundText
End Sub

Private Sub LocationDBCombo_GotFocus()
    '---------------------------------------------------------------------------------------------------
    ' Execute when Location ComboBox object receives the focus.
    '---------------------------------------------------------------------------------------------------
    On Error GoTo LocationDBComboGotFocusError
    ' Add, Delete, or Update Location
    piEditAddDelete = 2

    If (pbBlindUser) Then
      If (Len(Trim(LocationDBCombo.Text)) = 0) Then
        SpeakText "Location"
      Else
        SpeakText Trim(LocationDBCombo.Text)
      End If
    End If
    Exit Sub

LocationDBComboGotFocusError:
    ErrMsg ("LocationDBCombo GotFocus")
    Exit Sub
End Sub

Private Sub LocationDBCombo_KeyDown(KeyCode As Integer, Shift As Integer)
    '---------------------------------------------------------------------------------------------------
```

```
  ' Execute when the user presses (KeyDown) or releases (KeyUp)
  '-------------------------------------------------------------------------------------------------
  ' Save the Phone's physical location ID
  Select Case KeyCode
    Case vbKeyDown, vbKeyPageDown
      psLocationID = LocationDBCombo.BoundText
      If ((pbBlindUser) And (Len(Trim(LocationDBCombo.Text)) > 0)) Then
        SpeakText Trim(LocationDBCombo.Text)
      End If
    Case Else
  End Select
End Sub

Private Sub LocationDBCombo_KeyUp(KeyCode As Integer, Shift As Integer)
  '-------------------------------------------------------------------------------------------------
  ' Execute when the user presses (KeyDown) or releases (KeyUp)
  '-------------------------------------------------------------------------------------------------
  ' Save the Phone's physical location ID
  Select Case KeyCode
    Case vbKeyUp, vbKeyPageUp
      psLocationID = LocationDBCombo.BoundText
      If ((pbBlindUser) And (Len(Trim(LocationDBCombo.Text)) > 0)) Then
        SpeakText Trim(LocationDBCombo.Text)
      End If
    Case Else
  End Select
End Sub

Private Sub mnuAbout_Click(Index As Integer)
  '-------------------------------------------------------------------------------------------------
  ' Open the About form
  '-------------------------------------------------------------------------------------------------
  On Error GoTo mnuAboutError
  Select Case Index
    Case 0
      fAbout.Show vbModal
    Case Else
  End Select

  Exit Sub

mnuAboutError:
  ErrMsg ("About Menu")
  Exit Sub
End Sub
```

```
Private Sub mnuFileExit_Click(Index As Integer)
  '-------------------------------------------------------------------------------------------------------
  ' Exit Phone system when the Exit menu item is clicked
  '-------------------------------------------------------------------------------------------------------
  On Error GoTo mnuFileExitError

  Dim sMSPhoneTitle As String
  Dim lhWnd As Long

  Select Case Index
    Case 0
      ' Close MS Phone if active
      sMSPhoneTitle = "Phone"
      lhWnd = FindWindow(vbNullString, sMSPhoneTitle)
      If (lhWnd <> 0) Then
        plReturnValue = SetActiveWindow(lhWnd)
        plReturnValue = PostMessage(lhWnd, &H10, 0, 0&)
        lhWnd = FindWindow(vbNullString, "Phone")
        ' Select yes to Exit MS Phone
        If (lhWnd <> 0) Then
          plReturnValue = SetActiveWindow(lhWnd)
          SendKeys "{ENTER}", True
        End If
      End If
      Set fPhone = Nothing
      Set poVoiceText = Nothing
      ' Terminates execution
      End
    Case Else
  End Select

  Exit Sub

mnuFileExitError:
  ErrMsg ("File Exit Menu")
  Exit Sub
End Sub

Private Sub mnuOptionsBlindUser_Click(Index As Integer)
  '-------------------------------------------------------------------------------------------------------
  ' Blind user menu item that updates the Registry entry
  '-------------------------------------------------------------------------------------------------------
  On Error GoTo mnuOptionBlindUserError
  Select Case Index
```

```
    Case 0
      If (mnuOptionsBlindUser(0).Checked) Then
        mnuOptionsBlindUser(0).Checked = False
        pbBlindUser = False
        ' Disable Voice Text
        poVoiceText.Enabled = False
        Call VoiceTextErrorCheck(pciVTXT_ATTRIB_DISABLE)
      Else
        mnuOptionsBlindUser(0).Checked = True
        pbBlindUser = True

        Set poVoiceText = Nothing

        ' Initialize Voice Text
        pbRegisterFirstTime = True

        ' Set the speed at which speech is spoken (Words per Minute)
        ' Mininum Speech Speed = 50
        ' Maximum Speech Speed  = 250
        plSpeechSpeed = 75

        ' Registers an application with the Voice-Text object.
        RegisterVoiceText
        ' Enable Voice Text
        poVoiceText.Enabled = True
        Call VoiceTextErrorCheck(pciVTXT_ATTRIB_ENABLE)
      End If
      PutRegBoolean "HKEY_LOCAL_MACHINE\Software\BlindPhone", "BlindUser",
mnuOptionsBlindUser(0).Checked, True

    Case Else
  End Select

  Exit Sub

mnuOptionBlindUserError:
  ErrMsg ("Option Blind User Menu")
  Exit Sub
End Sub

Private Sub NameDBCombo_Click(Area As Integer)
  '---------------------------------------------------------------------------------------------
  ' Execute when the user Clicks on the NameDBCombo
  '---------------------------------------------------------------------------------------------
  ' Selected Person's ID
```

```
    psNameID = NameDBCombo.BoundText
End Sub

Private Sub NameDBCombo_GotFocus()
  '---------------------------------------------------------------------------------------------------
  ' Execute when Name ComboBox object receives the focus.
  '---------------------------------------------------------------------------------------------------
  On Error GoTo NameDBComboGotFocusError
  ' Add, Delete, or Update Name
  piEditAddDelete = 1

  If (pbBlindUser) Then
    If (Len(Trim(NameDBCombo.Text)) = 0) Then
      SpeakText "Name"
    Else
      SpeakText Trim(NameDBCombo.Text)
    End If
  End If

  Exit Sub

NameDBComboGotFocusError:
  ErrMsg ("NameDBCombo")
  Exit Sub
End Sub

Private Sub NameDBCombo_KeyDown(KeyCode As Integer, Shift As Integer)
  '---------------------------------------------------------------------------------------------------
  ' Execute when the user presses (KeyDown) or releases (KeyUp)
  '---------------------------------------------------------------------------------------------------
  ' Selected Person's ID
  Select Case KeyCode
    Case vbKeyDown, vbKeyPageDown
      psNameID = NameDBCombo.BoundText
      If ((pbBlindUser) And (Len(Trim(NameDBCombo.Text)) > 0)) Then
        SpeakText Trim(NameDBCombo.Text)
      End If
    Case Else
  End Select
End Sub

Private Sub NameDBCombo_KeyUp(KeyCode As Integer, Shift As Integer)
  '---------------------------------------------------------------------------------------------------
  ' Execute when the user presses (KeyDown) or releases (KeyUp)
  '---------------------------------------------------------------------------------------------------
```

```
  ' Selected Person's ID
  Select Case KeyCode
    Case vbKeyUp, vbKeyPageUp
      psNameID = NameDBCombo.BoundText
      If ((pbBlindUser) And (Len(Trim(NameDBCombo.Text)) > 0)) Then
        SpeakText Trim(NameDBCombo.Text)
      End If
    Case Else
  End Select
End Sub

Private Sub UpdateBtn_Click()
  '-----------------------------------------------------------------------------------------------
  ' Update a Name, Location, or a Phone Number
  '-----------------------------------------------------------------------------------------------
  On Error GoTo UpdateBtnClickError
  Select Case piEditAddDelete
    Case 1   ' Update Name
      If (Val(psNameID) > 0) Then
        plNameID = Val(psNameID)
        Set pQueryDef = pDBase.QueryDefs("Update Name")
        pQueryDef.Parameters("@Name") = Trim(NameDBCombo.Text)
        pQueryDef.Parameters("@NameID") = plNameID
        pQueryDef.Execute dbFailOnError
        pQueryDef.Close
        Set pQueryDef = Nothing
        NameData.Refresh
        NameDBCombo.BoundText = plNameID
        psNameID = NameDBCombo.BoundText
        piResponse = Alert("Name was updated Successfully.", "Information",
vbOKOnly + vbInformation, pbBlindUser, True)
      Else
        piResponse = Alert("Invalid selected person's name.", "Error", vbOKOnly +
vbCritical, pbBlindUser, True)
      End If
    Case 2   ' Update Location
      If (Val(psLocationID) > 0) Then
        plLocationID = Val(psLocationID)
        Set pQueryDef = pDBase.QueryDefs("Update Location")
        pQueryDef.Parameters("@Location") = Trim(LocationDBCombo.Text)
        pQueryDef.Parameters("@LocationID") = plLocationID
        pQueryDef.Execute dbFailOnError
        pQueryDef.Close
        Set pQueryDef = Nothing
        LocationData.Refresh
```

```
        LocationDBCombo.BoundText = plLocationID
        psLocationID = LocationDBCombo.BoundText
        piResponse = Alert("Location was updated Successfully.", "Information",
vbOKOnly + vbInformation, pbBlindUser, True)
      Else
        piResponse = Alert("Invalid selected phone's physical Location.", "Error",
vbOKOnly + vbCritical, pbBlindUser, True)
      End If
    Case 3  ' Update Phone
      If ((Val(psNameID) > 0) And (NameDBCombo.MatchedWithList)) Then
        If ((Val(psLocationID) > 0) And (LocationDBCombo.MatchedWithList)) Then
          Set pQueryDef = pDBase.QueryDefs("Get Phone ID And Number")
          pQueryDef.Parameters("@NameID") = Val(psNameID)
          pQueryDef.Parameters("@LocationID") = Val(psLocationID)
          Set pRsPhone = pQueryDef.OpenRecordset
          pQueryDef.Close
          Set pQueryDef = Nothing
          If (Not pRsPhone.EOF) Then
            plPhoneID = pRsPhone("PhoneID")
            Set pQueryDef = pDBase.QueryDefs("Update Phone Number")
            pQueryDef.Parameters("@Phone") = Trim(Display.Text)
            pQueryDef.Parameters("@PhoneID") = plPhoneID
            pQueryDef.Execute
            pQueryDef.Close
            Set pQueryDef = Nothing
            piResponse = Alert(NameDBCombo.Text & "'s " & LocationDBCombo.Text
& " phone number was updated to " & Display.Text & " Successfully.", "Information",
vbOKOnly + vbInformation, pbBlindUser, True)
          Else
            Display.Text = ""
            piResponse = Alert(NameDBCombo.Text & "'s " & LocationDBCombo.Text
& " phone number does not exist, Please use the add.", "Error", vbOKOnly + vbCritical,
pbBlindUser, True)
          End If
          pRsPhone.Close
          Set pRsPhone = Nothing
          Display.SetFocus
        Else
          piResponse = Alert("Invalid selected phone's physical location.", "Error",
vbOKOnly + vbCritical, pbBlindUser, True)
          LocationDBCombo.SetFocus
        End If
      Else
        piResponse = Alert("Invalid selected person's name.", "Error", vbOKOnly +
vbCritical, pbBlindUser, True)
```

```
        NameDBCombo.SetFocus
      End If
    Case Else
      piResponse = Alert("To update a phone number, please set focus on the Phone
Display EditBox.", "Information", vbOKOnly + vbInformation, pbBlindUser, True)
  End Select
  piEditAddDelete = 0

  Exit Sub

UpdateBtnClickError:
  ErrMsg ("UpdateBtnClick")
  Exit Sub
End Sub

Private Sub UpdateBtn_GotFocus()
  '---------------------------------------------------------------------------------------------
  ' Executes when the Dial Command Button object receives the focus by using the Tab
key.
  '---------------------------------------------------------------------------------------------
  On Error GoTo UpdateBtnGotFocus

    If ((pbBlindUser) And (GetAsyncKeyState(&H9) = 0 Or GetAsyncKeyState(&H9) =
-32767 Or GetAsyncKeyState(&H9) = 1)) Then
    SpeakText UpdateBtn.Caption
  End If

  Exit Sub

UpdateBtnGotFocus:
  ErrMsg ("UpdateBtnGotFocus")
  Exit Sub
End Sub
```

Module 5 VoiceText.Bas

```
'-----------------------------------------------------------------------------------------------
' Force explicit declaration of all variables in that module
'-----------------------------------------------------------------------------------------------
Option Explicit

Dim lErrNum As Long

Public Const pciVTXT_REGISTER As Integer = 1
Public Const pciVTXT_SPEAK As Integer = 2
Public Const pciVTXT_STOPSPEAK As Integer = 3
Public Const pciVTXT_PAUSE As Integer = 4
Public Const pciVTXT_RESUME As Integer = 5
Public Const pciVTXT_FASTFORWARD As Integer = 6
Public Const pciVTXT_REWIND As Integer = 7
Public Const pciVTXT_ATTRIB_ENABLE As Integer = 8
Public Const pciVTXT_ATTRIB_DISABLE As Integer = 9
Public Const pciVTXT_ATTRIB_SPEEDSET As Integer = 10
Public Const pciVTXT_CALLBACK As Integer = 11

Public Sub VoiceTextErrorCheck(VTXT_Cmd As Integer)
  '-----------------------------------------------------------------------------------------------
  ' Check Voice-Text Run-Time Errors.
  '-----------------------------------------------------------------------------------------------
  Dim sMessage As String

  lErrNum = Err.Number
  ' Run-Time error has occured.
  If lErrNum <> 0 Then
    Select Case VTXT_Cmd
      Case pciVTXT_REGISTER
        sMessage = "Register: "
      Case pciVTXT_SPEAK
        sMessage = "Speak: "
      Case pciVTXT_STOPSPEAK
        sMessage = "StopSpeak: "
      Case pciVTXT_PAUSE
        sMessage = "Pause: "
      Case pciVTXT_RESUME
        sMessage = "Resume: "
      Case pciVTXT_FASTFORWARD
        sMessage = "FastForward: "
      Case pciVTXT_REWIND
```

```
        sMessage = "Rewind: "
      Case pciVTXT_ATTRIB_ENABLE
        sMessage = "Enable VoiceText: "
      Case pciVTXT_ATTRIB_DISABLE
        sMessage = "Disable VoiceText: "
      Case pciVTXT_ATTRIB_SPEEDSET
        sMessage = "Set speed: "
      Case pciVTXT_CALLBACK
        sMessage = "Callback: "
    End Select

    piResponse = Alert(sMessage & "Error #" & Str(Err.Number) & " was generated by "
& Err.Source & " - " & Err.Description, "Error", vbOKOnly + vbCritical, pbBlindUser,
True)

    lErrNum = 0
  Else
    ' No Run-Time error, display status message.
    Select Case VTXT_Cmd
      Case pciVTXT_REGISTER
        sMessage = "VoiceText Registered"
      Case pciVTXT_SPEAK
        sMessage = "Speech started"
      Case pciVTXT_STOPSPEAK
        sMessage = "Speech stopped"
      Case pciVTXT_PAUSE
        sMessage = "Speech paused"
      Case pciVTXT_RESUME
        sMessage = "Speech resumed"
      Case pciVTXT_FASTFORWARD
        sMessage = "Fastforwarding"
      Case pciVTXT_REWIND
        sMessage = "Rewinding"
      Case pciVTXT_ATTRIB_ENABLE
        sMessage = "VoiceText enabled"
      Case pciVTXT_ATTRIB_DISABLE
        sMessage = "VoiceText disabled"
      Case pciVTXT_ATTRIB_SPEEDSET
        sMessage = "Speed set"
      Case pciVTXT_CALLBACK
        sMessage = "Callback notification initiated"
    End Select
  End If
End Sub
```

```
Function SpeakingDone()
  '-----------------------------------------------------------------------------------------------
  ' Text-To-Speech Callback speaking done notification function.
  '-----------------------------------------------------------------------------------------------
  fPhone.StatusBar.Panels(1).Text = "Speaking Done notification"
End Function

Function SpeakingStarted()
  '-----------------------------------------------------------------------------------------------
  ' Text-To-Speech Callback speaking started notification function.
  '-----------------------------------------------------------------------------------------------
  fPhone.StatusBar.Panels(1).Text = "Speaking Started notification"
End Function
```

APPENDIX D

WHAT TO SAY? VOICE COMMANDS

Voice commands could be used to simulate pressing a key or a combination of keys. A user could say any of the words in the right column of Table I Page 85 which would be the same as pressing the key on the keyboard. The "Press" word has to proceed the Alt, Control, Shift, and Control + Alt or any of the keyboard keys in Table II Page 87 whenever they are used in conjunction with any of the contents of Table I Page 85 (i.e. "Press Alt F4"). The command "Press Capital ?" would capitalize any of the alphabets in Table I Page 85 by replacing the question mark with a letter.

Table I

LIST OF PRERECORDED LETTERS AND NUMBERS

LETTERS AND NUMBERS	WHAT TO SAY
0	Zero
1	One
2	Two
3	Three
4	Four
5	Five
6	Six
7	Seven
8	Eight
9	Nine
a	Alpha
b	Bravo
c	Charlie
d	Delta

e	Echo
f	Fox
g	Golf
h	Hotel
I	India
j	Juliet
k	Kilo
l	Leema
m	Mike
n	November
o	Oscar
p	Papa
q	Quebec
r	Romeo
s	Sierra
t	Tango
u	Uniform
v	Victor
w	Whiskey
x	X-ray
y	Yankee
z	Zulu

Table II

OTHER KEYBOARD KEYS

Other Keyboard keys	What To Say
F1	F One
F2	F Two
F3	F Three
F4	F Four
F5	F Five
F6	F Six
F7	F Seven
F8	F Eight
F9	F Nine
F10	F Ten
F11	F Eleven
F12	F Twelve
Esc	Escape
~	Tilde
!	Exclamation Mark
@	At Symbol
#	Number Sign
$	Dollar Sign
%	Percent
^	Caret
&	And
*	Asterisk
(	Left Parenthesis
)	Right Parenthesis
_	Underline
-	Dash or Minus
+	Plus
=	Equal
Backspace	Backspace
Print Screen	Print Screen
Insert	Insert
Home	Home
Page Up	Page Up
Delete	Delete
End	End
Page Down	Page Down
Move Up	Up Arrow

Move Down	Down Arrow
Move Right	Right Arrow
Move Left	Left Arrow
Tab	Tab
{	Left Curly Brace
}	Right Curly Brace
[	Left Bracket
]	Right Bracket
|	Vertical Bar
\	Back-Slash
:	Colon
;	Semi-Colon
“	Quote
‘	Apostrophe
Num. Lock	Num. Lock
Enter or return	Enter
<	Less Than
>	More Than
?	Question Mark
,	Comma
.	Period
/	Slash
Space	Space

Table III

BLIND PHONE SYSTEM COMMANDS

Phone System Commands	Description
Add	Add a Name, Location, or phone number depending on which control has focus
Close window	Exit the most active application
Delete	Delete a Name, Location, or phone number depending on which control has focus
Dial	Activate MS Phone and then dial the phone number
Display	Set focus on the phone display EditBox
Done/Hang Up	Release the phone line
Get Phone	Retrieve the phone number from the database
Location	Set focus on the location ComboBox
Name	Set focus on the name ComboBox
Next	Move focus to the next control
Previous	Move focus to the previous control
Read Selection	Convert the Highlighted text to speech
Select	Highlight text
Update	Update the Name, Location, or phone number depending on which control has focus

VITA

Nabil Amine Nabhan

Candidate for the Degree of

Master of Science

Thesis: PROVIDING COMPUTER TELEPHONY TO DISABLED USERS WITH VOICE RECOGNITION

Major Field: Computer Science

Biographical:

Personal Data: Born in Hawali, Kuwait, On Friday, January 15, 1965, the son of Amine and Majeda Nabhan.

Education: Graduated from AL-Ikhlas High School, Al-Jabria, Kuwait in May 1983; attended Tulsa University English Institute; attended Rogers State College; received Associate of Science degree in Electronics Engineering Technology, Bachelor of Science degree in Electronics Engineering Technology, and Bachelor of Science degree in Computer Science from Oklahoma State University, Stillwater, Oklahoma in May 1987, July 1988, and May 1991, respectively.
Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in May 1997.

Experience: Raised in Hawali, Kuwait; employed as a mechanic during summers; employed by Oklahoma State University, Electronics Engineering Technology Department as a grader and as a teaching assistance; employed by Consulting Engineers as CAD Operator; employed by dTech Group, Inc. as a Programmer/Analyst; employed by Windows Technology Group, Inc. as a Software Engineer/Consultant, 1991 to present.

Professional Memberships: IEEE and Computer Society, Phi Kappa Phi Honor Society, and Golden Key Honor Society.